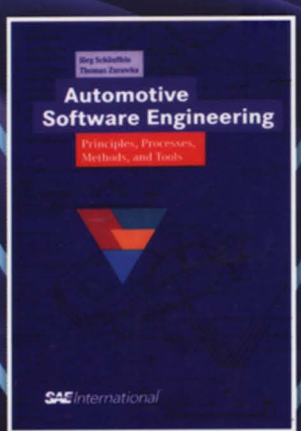
汽车软件工程

——原理·过程·方法·工具

Automotive Software Engineering
Principles, Processes, Methods, and Tools

[德] Jörg Schäuffele
Thomas Zurawka 著
张 聚 等译



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

汽车软件工程

——原理·过程·方法·工具

Automotive Software Engineering
Principles, Processes, Methods, and Tools

『德』Jörg Schäuffele 著
『德』Thomas Zurawka
张 聚 等译

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书系统地阐述汽车电子系统和软件开发的原理、过程、方法和工具。全书内容分为三部分。第一部分是相关基础知识，包括控制系统、嵌入式系统、实时系统、分布和网络控制系统，以及系统的安全性和可靠性等。第二部分是汽车软硬件系统开发的支持过程和核心开发过程，支持过程包括配置管理、项目管理、供应商管理、需求管理和质量管理等；核心开发过程包括汽车软件系统的需求分析、设计、实现、集成和测试等。第三部分介绍汽车软件设计方法与工具，以及生产和服务过程中涉及的工具。

书中采用大量的图表和实例阐述相关的理论并提供工程背景知识，具有很强的实践性和实用性。本书可供从事汽车电子和软件系统研究、开发、管理与维护的工程技术人员使用和参考，也可作为高等院校相关专业本科生和研究生的教材或参考书。

Translated from the English language edition:
Automotive Software Engineering: Principles, Processes, Methods, and Tools by Jörg Schäuffele and Thomas Zurawka
Copyright ©2005 by Society of Automotive Engineers International (SAE International), 400 Commonwealth Drive, Warrendale, PA 15096-001, United States America (USA)
ISBN: 0-7680-1490-5

Chinese language edition Copyright ©2007 by Publishing House of Electronics Industry.
All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without prior permission in writing from Publishing House of Electronics Industry

本书简体中文版由电子工业出版社出版。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。

版权贸易合同登记号　图字：01-2007-0518

图书在版编目（CIP）数据

汽车软件工程：原理·过程·方法·工具 /（德）绍弗勒（Schäeuffele,J.），（德）卢拉弗卡（Zurawka,T.）著；张聚等译. —北京：电子工业出版社，2008.1
书名原文：Automotive Software Engineering：Principles，Processes，Methods，and Tools
ISBN 978-7-121-05475-4

I. 汽…　II. ①绍…　②卢…　③张…　III. 汽车—电子系统—系统设计　IV. U463.6

中国版本图书馆 CIP 数据核字（2007）第 184043 号

责任编辑：朱清江
印　　刷：北京市顺义兴华印刷厂
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编　100036
开　　本：787×1092　1/16　印张：16　字数：350 千字
印　　次：2008 年 1 月第 1 次印刷
印　　数：4000 册　　定价：38.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

名家导读——

软件技术在汽车中的作用

通过应用新的开发方法处理复杂性问题

没有一种其他的技术像软件技术那样能够在设计过程中提供这么大的灵活性。汽车功能的增加，汽车中控制系统的网络化，汽车高可靠性和安全性的要求，以及汽车型号系列的增加，为软件技术在汽车中的应用提供了巨大的舞台。解决汽车中的这些高要求，对于汽车整体生产商和汽车部件的提供商是个很大的挑战：在开发阶段必须采取措施来保证软件和系统中与汽车安全性相关的功能的实现。

——**Siegfried Dais 博士**

副董事长，博世（Bosch）集团（德国，斯图加特）

软件正成为汽车中的一种关键性的产品

汽车中的软件部分越来越成为汽车制造商的一种关键性的产品。在汽车领域，电子和软件部分已经成为汽车技术改进和革新的一种主要推动力。汽车中约 90%的革新通过电子和软件技术来完成。一方面，汽车中越来越多的典型功能转由软件来实现；另一方面，通过网络化那些原来是分离的功能，使得开发一些全新的汽车功能成为可能。系统工程方法和技术的综合应用是成功开发汽车整体系统的至关重要的因素。本书讨论汽车软件工程方面的问题，并对主要应用软件的开发进行了着重讲述。

——**Hans-Georg Frischkorn**

系统体系和集成副总裁，宝马（BMW）集团（德国，慕尼黑）

从成本驱动到竞争优势

汽车工业中的技术飞跃只有通过软件技术的进步才能达到。在系统开发中不同专业领域工程师之间的富有成效的合作需要一种统一的背景知识，通用一致的概念和一种合适的开发过程模型。借助于工程实际中的实例，本书阐述相关的基本内容和方法。

——**Wolfgang Runge 博士**

首席执行官（CEO），采埃孚（ZF）转向系统股份有限公司（德国）

嵌入式系统是汽车的重要组成部分

在竞争日益激烈的汽车业务领域，嵌入式系统为区别不同的汽车产品和服务提供了机会。根据摩尔定律，电子产品的功能变得越来越强大而价格却持续在下降。为了能够处理由此而得到的系统的复杂性，必须对本书中所讨论的一些原理、应用、方法和工具有深入的了解。

我很赞赏 Zurawka 博士和 Jörg Schäuffele 先生在这个非常重要的领域所开展的系统性的工作。对于所有希望加入到汽车行业中的企业来说，了解本书的内容是必要的。由于我们将不断地进一步改善工程开发过程，我们期待着能够得到该领域读者的更多帮助。

——Craig A.Brown

通用电气工程部，通用汽车公司（美国，底特律）

高等教育的机会和挑战

正是在斯图加特地区，汽车制造业是支柱产业。一些著名的汽车制造商和零部件供应商的研发中心为斯图加特地区提供了大量的就业岗位。在斯图加特大学，软件技术课程是工科专业学生必不可少的学习部分。本书为学生提供了接触和了解汽车工业中软件开发和应用的实践经验，且本书中介绍的软件开发方法甚至可以作为模型，在其他工业领域中参考应用。

——Peter Göhner 博士、教授

斯图加特大学自动化与软件技术研究所（德国，斯图加特）

汽车开发需要一种基于集成模型的新方法

汽车开发技术进步的自然结果是，必须满足用户的需求并与政府的相关规定保持一致。与这些方面相关的问题正成为汽车电子系统的特殊领域。事实上，在汽车中当前的绝大部分的竞争优势在于汽车中的电子系统，而软件系统又是汽车电子系统的重要组成部分。安全、可靠、低成本和快速地开发基于软件的汽车电子系统是汽车制造商和零部件提供商所关心的重要问题。本书从一个集成模型的角度，讨论了汽车电子系统开发这个复杂的问题，为学生们提供了解决汽车嵌入式软件工程问题的一种集成的系统解决模式和方案。本书为汽车软件集成化开发的实际工作提供了基础。

Mark Thompson 教授

电气和计算机工程学院，凯特林（Kettering）大学（美国）

需要一种新的系统科学

由于硬件的开发成本和制造成本不断且急剧地增加，以及对于产品需求的变化越来越快，嵌入式系统、子系统或者集成电路的设计者开始把目光转向借助软件技术作为处理这些问题的一种方法。然而这种转变也带来一系列新挑战。例如，用于交通系统的软件技术必须满足一些由于具体的软件实现平台和环境而引起的约束条件，因而使得传统的软件技术不能适用。这也就毫不奇怪，大约 30%的汽车中的严重故障是由于软件的错误而造成的。我们需要一种新的系统科学以统一的方式来处理软件抽象和真实的物理世界间的联系。本书是作者在汽车软件开发的方法论和开发工具方面所作的开创性工作的总结。强调方法论方面的重要性一点儿也不为过，因为开发工具的作用只有通过合适的方法才能发挥出来,这些是过去几年我在电子设计自动化方面工作中得到的经验教训。本书的独特价值是成功地把方法与工具有机地结合起来。

——Alberto Sangiovanni Vincentelli 教授
加利福尼亚大学电子工程学院（美国，伯克利）
EDA 先驱、EECS 主席
美国 Cadence 公司创办人之一、技术总顾问

本书的读者对象

本书可供所有在汽车或者汽车零部件的开发、生产和服务领域工作的人员，以及从事与汽车软件相关的行业的人员使用。我们希望本书能够为他们提供一些实用的帮助。

此外，本书也可作为相关专业的工科大学生学习使用，也可用于汽车行业和相关领域新的从业人员入门学习和培训使用。掌握自动控制和系统理论方面的基本知识对于学习本书是有益的，但并不是必须的。

读者很可能希望对于本书中的某些章节有系统深入的阐述，我们欢迎并非常高兴读者能够对于本书的内容提出意见和改进建议，并与我们进行深入的交流。

致谢

借此机会我们想表达对于我们所有客户的谢意，感谢他们多年来与我们进行的成功和相互信任的合作。如果没有与他们进行富有价值的交流和合作，本书是不可能写成的。

感谢宝马集团的友好支持，允许在本书中阐述我们在宝马集团的开发项目中（本书的第一作者曾为宝马集团的员工）所获得的经验和体会。感谢 Hans-Georg Frischkorn 先生为本书写了导言，感谢 Heinz Merkle 先生，Helmut Hochschwarzer 博士，Maximilian Fuchs 博士，Dieter Nazareth 教授，以及他们所有的同事。

本书中的许多过程和方法，是在与博世公司的多年紧密和相互信任的合作中形成的。在此期间所建立的并得到广泛应用的开发过程在本书中的许多地方都能够找到。感谢博世公司汽车底盘系统、柴油驱动系统、汽油驱动系统以及科学研究部门的员工。

衷心感谢 Sigfried Dais 博士，Wolfgang Runge 博士，Craig A. Brown 先生，Peter Göhner 教授，Mark Thompson 教授，Alberto Sangiovanni Vincentelli 教授为本书写了导言（软件技术在汽车中的作用）。

感谢我们所有的同事在过去的几年中、在许多方面对于本书所做的贡献。

Jörg Schäuffele, Thomas Zurawka
德国斯图加特
2005 年 6 月

前　言

经过100多年的发展历史后，汽车作为一种产品仍然继续以一种很快的速度在演化发展。自从20世纪70年代早期以来，汽车中电子系统和软件的持续增加，对于汽车的发展产生重大的影响，并且这种趋势还在继续。这种发展趋势的结果是汽车的开发、生产和服务的方式正在发生一种根本性的改变。应用软件技术来实现汽车的功能为开发者解决现有的一些复杂矛盾问题提供了更大的自由度和新的解决方法。由此而产生的汽车系统的复杂性只有通过应用合适的过程、方法和工具才能解决。

在过去的几年中，人们提出了多种方法和标准用于汽车电子系统的软件开发。这些方法和标准统一以汽车软件工程这个术语来概括描述。

多年来，在汽车软件工程领域，存在着一系列复杂的概念和术语。从事汽车软件工程领域的人们经常接触这些术语和概念，但是我们有时很难准确理解这些术语和概念的准确定义和含义。事实上，有时同样的术语和概念应用在不同的场合，表达的含义是不同的。例如，“Process”这个术语不仅出现在控制工程的领域，而且也应用于与实时系统相关的情况，更不用说也应用于产品的开发过程。在这些应用场合，“Process”所指的含义是完全不同的。在本书中，我们将首先定义一些最为基本的概念，以便于随后章节的统一应用。

本书中的章节重点是介绍汽车软件系统开发的过程、方法和工具，同时也强调软件开发和汽车系统工程之间的相互作用。本书中阐述的开发过程具有模型化的特点，也就是说，开发过程是实际工作的一种抽象、理想化的图形表示。虽然开发过程模型可以为各种项目的开发提供指导，但具体到某个实际的开发项目，必须对开发过程模型的适用性做出评价，如有可能还需做适当的修改。在本书中，我们将努力对开发的过程、方法和工具做一个清楚准确的描述。

由于我们讨论的问题涉及的面比较广，不可能对于所有的主题进行深入的讨论，因而我们只限于讨论与汽车工业相关的领域。

我们不能保证在本书中给出的是唯一合适的或唯一一种完整的汽车软件开发方法。然而作为ETAS的员工，我们确信我们公司提供的工具和产品非常适合用于支持本书中介绍的开发过程和方法。

来自工程实际的例子

只有成功地解决了工程实际中的大量问题，一种开发过程模型才被认为是一支开发团队的成功工具。本书并不是一本理论性的教科书，本书中所有的建议和观点都是基于实际的应用例子，并通过具体例子来形象地叙述。同时也考虑到，我们在过去的几年中与汽车制造商和汽车零部件供应商之间的密切合作所获得的一些经验和体会。例如，包括生产和服务方面的产品开发过程以及在研究项目方面所获得的经验。

译者前言

汽车工业是现代工业的典型代表，一个国家汽车工业的发展水平一定程度上代表一个国家的现代化水平。在当今工业水平高度发达的国家，如美国、日本和德国，无一不是汽车工业的强国。汽车工业的生产方式和管理模式对现代制造业的发展具有巨大的促进和推动作用。正是因为发展汽车工业的重大意义，国家非常重视汽车工业，并力争使汽车产业在2010年前发展成为国民经济的支柱产业。

快速发展的中国汽车工业正在推动整个国家汽车电子产业的技术升级，节能减排、汽车电子智能化、动力总成、安全管理、通信导航、信息娱乐及车载网络等成为汽车领域热门的话题，并为软件技术在汽车中的应用提供了巨大的舞台。在汽车领域，电子和软件部分已经成为汽车技术改进和革新的一种主要推动力。应用软件技术来实现汽车的功能为开发者解决现有的一些复杂问题提供了更大的自由度和新的解决方法。然而，汽车系统的复杂性只有通过应用合适的开发过程、开发方法和工具才能解决。

本书系统地阐述汽车电子系统和软件开发的过程、方法和工具。该书的主要内容可以归纳为三部分：（1）汽车软件设计与开发的基础知识。包括控制系统、离散系统、嵌入式系统、实时系统、分布和网络控制系统，以及系统的安全性和可靠性等基本知识。（2）汽车软硬件系统开发的外部支持过程和核心开发过程。外部支持过程包括配置管理、项目管理、供应商管理、需求管理和质量管理等内容。核心开发过程即汽车电子系统设计与开发的详细过程，包括汽车软件系统的需求分析、系统的体系结构、系统的设计、系统的实现、系统的集成和测试等内容。（3）汽车软件设计的方法与工具以及生产和服务过程中涉及的工具。书中还列举了大量的图表和实例用于阐述相关的方法和提供工程背景知识。

本书不仅仅面向那些从事汽车研究、开发、管理与维护的人员，广大的工科类在校大学生（包括机械、车辆、电气、电子、自动化、计算机、信息、软件以及系统与管理类专业的学生）也将从这本书中获益。这也是德国斯图加特大学的教授极力向工科大学生推荐该书的原因。

参与本书翻译工作的有：张聚博士，徐新黎博士，张霓博士，汤丽平博士，王槊华，杨光利，刘江。全书翻译工作由张聚统稿、审校和安排组织完成。在翻译过程中，译者虽然力求准确和风格一致，但限于学识和水平，译文中难免有不当之处，欢迎读者批评指正。

张　聚

2007年8月于杭州

目　录

第 1 章　引言与概述 …… (1)
1.1　驾驶员－车－环境系统 …… (2)
1.1.1　汽车电子系统的构成和工作原理 …… (2)
1.1.2　汽车电子系统和环境 …… (4)
1.2　汽车电子系统概述 …… (5)
1.2.1　驱动部分电子系统 …… (6)
1.2.2　底盘电子系统 …… (8)
1.2.3　车身电子系统 …… (10)
1.2.4　多媒体系统 …… (11)
1.2.5　分布式网络电子系统 …… (12)
1.2.6　小结和展望 …… (13)
1.3　逻辑体系结构概述 …… (13)
1.3.1　汽车中的 ECU 网络和功能网络 …… (13)
1.3.2　开环/闭环控制和监视系统的逻辑体系结构 …… (14)
1.4　汽车的开发过程 …… (15)
1.4.1　汽车开发概述 …… (15)
1.4.2　电子系统开发概述 …… (15)
1.4.3　电子系统和软件开发的核心过程 …… (18)
1.4.4　电子系统和软件开发的支持过程 …… (20)
1.4.5　电子系统和软件的生产及服务 …… (21)
1.5　电子系统软件开发方法和工具 …… (22)
1.5.1　基于模型的开发 …… (22)
1.5.2　综合质量管理 …… (24)
1.5.3　减少开发风险 …… (25)
1.5.4　标准化和自动化 …… (26)
1.5.5　汽车开发步骤 …… (28)
第 2 章　基础知识 …… (29)
2.1　开环和闭环控制系统 …… (29)
2.1.1　建模 …… (29)
2.1.2　结构图 …… (30)
2.2　离散系统 …… (33)
2.2.1　离散时间信号和系统 …… (34)

2.2.2 离散值信号和系统 …… (35)
2.2.3 离散时间且离散值信号与系统 …… (35)
2.2.4 状态机 …… (36)
2.3 嵌入式系统 …… (37)
2.3.1 微控制器结构 …… (38)
2.3.2 存储技术 …… (39)
2.3.3 微控制器编程 …… (41)
2.4 实时系统 …… (45)
2.4.1 定义任务 …… (46)
2.4.2 实时性需求的定义 …… (47)
2.4.3 任务的状态 …… (49)
2.4.4 处理器的调度策略 …… (50)
2.4.5 实时操作系统的构成 …… (53)
2.4.6 任务间的相互作用 …… (54)
2.5 分布式网络系统 …… (58)
2.5.1 系统的逻辑体系结构和技术体系结构 …… (60)
2.5.2 逻辑通信链路定义 …… (61)
2.5.3 技术网络拓扑结构定义 …… (62)
2.5.4 报文定义 …… (63)
2.5.5 通信网络管理组织结构 …… (64)
2.5.6 总线仲裁机制 …… (67)
2.6 系统可靠性、安全性、监控和诊断 …… (68)
2.6.1 基本概念 …… (69)
2.6.2 系统的可靠性和可用性 …… (70)
2.6.3 系统的安全性 …… (73)
2.6.4 系统监控和诊断 …… (75)
2.6.5 电控单元监控系统的构成 …… (79)
2.6.6 电控单元诊断系统的构成 …… (81)
2.7 小结 …… (85)
第 3 章 电子系统和软件开发的支持过程 …… (87)
3.1 系统论的基本定义 …… (87)
3.2 过程模型和标准 …… (90)
3.3 配置管理 …… (91)
3.3.1 产品生命周期 …… (91)
3.3.2 型号系列 …… (91)
3.3.3 版本和配置 …… (92)
3.4 项目管理 …… (94)

3.4.1 项目计划 …… (94)
3.4.2 项目跟踪和风险管理 …… (98)
3.5 供应商管理 …… (98)
3.5.1 系统及其组成部分的责任承担者 …… (99)
3.5.2 规格说明和集成的接口 …… (99)
3.5.3 开发过程的交叉合作定义 …… (99)
3.6 需求管理 …… (100)
3.6.1 用户需求挖掘、记录和解释 …… (101)
3.6.2 用户需求跟踪 …… (104)
3.7 质量保证 …… (105)
3.7.1 集成和测试程序 …… (105)
3.7.2 软件质量保证方法 …… (106)
第 4 章 电子系统和软件的核心开发过程 …… (107)
4.1 要求和限制条件 …… (108)
4.1.1 系统和组件责任的分担 …… (108)
4.1.2 系统工程和软件工程的协调 …… (108)
4.1.3 基于模型的软件开发 …… (110)
4.2 基本概念 …… (110)
4.2.1 过程、处理步骤和产品 …… (110)
4.2.2 方法和工具 …… (111)
4.3 用户需求分析和系统逻辑结构的确定 …… (112)
4.4 逻辑功能结构分析和系统技术结构的确定 …… (114)
4.4.1 开环和闭环控制系统的分析和技术结构的确定 …… (117)
4.4.2 实时系统的分析和确定 …… (117)
4.4.3 分布式网络控制系统的分析和确定 …… (118)
4.4.4 可靠性及与安全相关系统的分析和规范 …… (118)
4.5 软件的需求分析和软件体系结构的确定 …… (119)
4.5.1 软件组件和组件间接口的描述 …… (120)
4.5.2 软件层次结构的描述 …… (121)
4.5.3 软件运行状态的描述 …… (122)
4.6 软件组件的描述 …… (123)
4.6.1 数据模型的描述 …… (124)
4.6.2 动态模型的描述 …… (124)
4.6.3 实时模型的描述 …… (126)
4.7 软件组件的设计和实现 …… (128)
4.7.1 考虑非功能性的产品特性要求 …… (128)
4.7.2 数据模型的设计和实现 …… (130)

4.7.3 动态模型的设计和实现 …… (130)
4.7.4 实时模型的设计和实现 …… (131)
4.8 软件组件的测试 …… (132)
4.9 软件组件的集成 …… (132)
4.9.1 程序版本和数据版本的产生 …… (133)
4.9.2 描述文件的产生 …… (134)
4.9.3 文档的产生 …… (134)
4.10 软件集成测试 …… (135)
4.11 系统组件的集成 …… (136)
4.11.1 软件和硬件的集成 …… (137)
4.11.2 ECU、设定点发生器、传感器和执行器的集成 …… (137)
4.12 系统的集成测试 …… (138)
4.13 校准 …… (141)
4.14 系统测试和验收测试 …… (141)
第5章 开发方法与工具 …… (143)
5.1 工具和电子控制单元之间的非车载接口 …… (144)
5.2 逻辑系统结构分析和技术系统结构描述 …… (145)
5.2.1 开环和闭环控制系统的分析和描述 …… (145)
5.2.2 实时系统的分析和描述 …… (148)
5.2.3 分布式网络化系统的分析和描述 …… (152)
5.2.4 可靠和安全系统的分析和描述 …… (155)
5.3 软件功能的描述和有效性验证 …… (160)
5.3.1 软件体系结构和软件组件的描述 …… (161)
5.3.2 数据模型描述 …… (164)
5.3.3 用框图描述行为模型 …… (164)
5.3.4 基于真值表的行为模型描述 …… (166)
5.3.5 行为模型的状态机描述 …… (168)
5.3.6 行为模型的高级语言描述 …… (171)
5.3.7 实时模型的描述 …… (172)
5.3.8 通过仿真和快速原型来验证描述 …… (173)
5.4 软件函数的设计及实现 …… (179)
5.4.1 考虑产品的非功能特性 …… (179)
5.4.2 定点和浮点算法的设计及实现 …… (185)
5.4.3 软件结构的设计和实现 …… (196)
5.4.4 数据模型的设计与实现 …… (199)
5.4.5 行为模型的设计与实现 …… (202)
5.5 软件功能的集成和测试 …… (204)

5.5.1 回路软件仿真 …… (205)
5.5.2 实验室汽车和试验台 …… (206)
5.5.3 实验性的、原型和产品车辆 …… (210)
5.5.4 实验的设计及自动化 …… (211)
5.6 软件功能的校准 …… (211)
5.6.1 离线和在线校准过程 …… (212)
5.6.2 通过 Flash 编程的软件更新 …… (213)
5.6.3 微控制器信号和仪器使用信号的同步测量 …… (214)
5.6.4 下载和评估车载诊断数据 …… (215)
5.6.5 离线校准 …… (215)
5.6.6 在线校准 …… (216)
5.6.7 在线校准的离线接口分类 …… (216)
5.6.8 CAL-RAM 管理 …… (220)
5.6.9 参量和数据版本管理 …… (222)
5.6.10 实验的设计和自动化 …… (223)
第 6 章 用于生产与服务的方法和工具 …… (224)
6.1 非车载诊断 …… (225)
6.2 软件功能参数设置 …… (225)
6.3 通过 Flash 编程的软件更新 …… (227)
6.3.1 Flash 存储器的擦除和编程 …… (227)
6.3.2 诊断界面的 Flash 编程 …… (227)
6.3.3 安全性要求 …… (228)
6.3.4 可用性要求 …… (230)
6.3.5 引导程序转换和 Flash 编程 …… (231)
6.4 汽车电子系统的启动和测试 …… (232)
第 7 章 总结和展望 …… (233)
参考文献 …… (235)

第 1 章

Chapter 1

引言与概述

为了满足如今日益增长的消费者需求以及更加合乎法律化的要求，现代汽车制造应基于以下两点来考虑：

（1）减少燃料消耗和有害气体的排放；

（2）增加驾驶安全性以及驾驶员和乘客的舒适度。

这些要求都已经和现代汽车电子产品的发展紧紧地联系在了一起。因此汽车已经成为当今技术要求最高、最综合的消费产品。然而同时值得注意的是，人们对于汽车电子产品的要求和对其他领域的电子产品的要求是截然不同的。以下列出了对于汽车电子产品最主要的要求：

（1）工作在苛刻的环境（温度、湿度、振动）或者是工作在对电磁兼容性有严格要求的环境；

（2）严格的可靠性要求和可用性要求；

（3）严格的安全性要求；

（4）有相对较长的产品生命周期。

尽管人们对于汽车电子产品的技术要求非常高，但是设计开发者依旧在如何降低成本，如何缩短产品开发周期以及如何处理大量的不同型号系列模型等方面面临巨大压力。然而在大批量生产时，这些非技术方面的需求必须得到满足。

为了成功开发一个随车携带电子产品项目，项目领导者必须一方面处理变得日益复杂的产品；另一方面在保持产品质量稳定的同时处理好风险与成本的控制问题。

对于想要在汽车电子系统的开发、生产以及服务方面建立合适可行的方法，及想要用面向工程应用的标准和工具来支持这些方法的人员来说，对汽车工程的要求和发展趋势有个基本的理解是必不可少的。本章将对于目前的发展现状做一个介绍性的分析，目的是为理解现代汽车电子技术未来的发展和面临的挑战提供必要的基础和背景知识。

本章首先对汽车电子系统以及它们的功能进行概况性的介绍，随后介绍用于汽车电子系统开发和汽车软件开发的方法。本章的最后对基于模型的工程学方法进行了介绍。

本书剩余章节详细讨论了相关的基础知识（第 2 章）、开发过程（第 3、4 章）、用于

开发汽车软件的方法和工具（第5章）和用于汽车软件生产及服务的方法和工具（第6章）。在本书中，特别强调了发动机、底盘和车身等汽车子系统。书中也提到了多媒体领域，但没有详细展开。第7章对于本书讨论的内容作了回顾和总结，同时也展望了汽车电子发展的未来远景和将面临的挑战。

1.1 驾驶员—车—环境系统

任何一个开发项目的目标都是致力于完成一项新的或者是改善已有的汽车功能。在本书中，我们所说的“功能”指的是所有的汽车功能特性。最终，用户（比如驾驶员）能直接或者间接地体验到这些功能所提供的价值或者带来的便利。

一个功能特征到底是通过机械、液压、电气还是通过电子系统来具体实现，对于用户来说，是次要的。

但是，对于考虑如何实现这些功能的工程师们来说，综合应用电子系统、机械系统、电气系统或者液压系统能带来很多的好处，特别是在考虑到可靠性、重量、安装的空间限制以及成本等方面。正是基于以上这些原因，汽车电子技术已成为实现汽车领域许多革新的一项关键技术。事实上在今天，汽车上几乎所有功能都在电子系统的控制或者监控之下。

1.1.1 汽车电子系统的构成和工作原理

下面通过一个关于汽车电子液压制动系统的例子，来具体了解汽车中电子系统的构成和工作原理。

举例：电液制动控制系统结构介绍[1]

图1-1展示了博世公司电子控制液压制动系统的结构。如图所示，电液制动系统由制动助力器（Brake Booster）功能、防抱死制动系统（Antilock Braking System，ABS）以及电子稳定程序（Electronic Stability Program，ESP）功能构成。

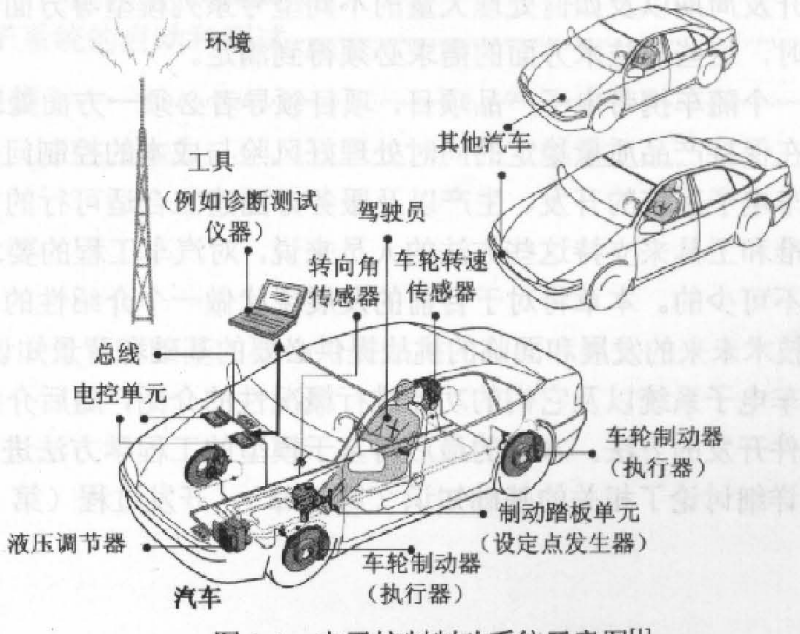


图1-1 电子控制制动系统示意图[1]

驾驶员进行制动操作时，踏板行程传感器探知驾驶员的制动意图，并把这一信息传给电子控制单元（Electronic Control Unit，ECU），ECU 综合踏板行程传感器（相当于制动系统的设定值）、轮速传感器、转向盘转向角传感器等各路信号，计算出 ECU 的输出变量值，随后传送给液压调节器，并进一步转化为车轮上制动装置的制动压力。车轮上制动装置直接影响汽车（被控对象）的驾驶状态，因此，车轮上制动装置被称为是执行器。

车载电子控制元件之间通过数据总线进行通信，例如 CAN 总线，因此一些较复杂的涉及多个 ECU 参与完成的功能也能够实现。其中的一个例子就是汽车牵引力控制系统（Traction Control System，TCS），它需要在汽车发动机操控和制动系统之间起协调作用。

事实上通过电子液压制动系统的结构，已经表明了汽车中所有的电子控制系统（开环/闭环）和汽车监控系统的典型系统构成。通常，一个控制系统包含以下的组成部分：设定点发生器、传感器、执行器、电子控制单元（ECU）和被控系统。电子控制元件间的网络化使数据交换更加便利。

在更高层次的驾驶员－车－环境系统中还包括驾驶员部分和环境部分，它们对于汽车的行为都能产生影响。

单单看 ECU 本身，它只是提供了实现功能的一种手段，是一个孤立的部分，对于汽车用户来说它并没有显而易见的价值。只有一个包含电子控制元件、设定点发生器、传感器和执行器的完整系统才能对被控对象起影响和监控作用，比如对用户的动作和请求做出回应。然而在许多情况下，特别是在嵌入式系统（实际系统经常如此）的情形，汽车功能的电子实现对于汽车用户来说甚至是不可见的。

如图 1-2 所示，车载控制和监控系统可以通过控制方框图来表示。系统的组成部分以方块表示，箭头表示方块之间信号的流向。有关控制和监视技术的基本知识和术语，见本书的 2.1 和 2.6 节。

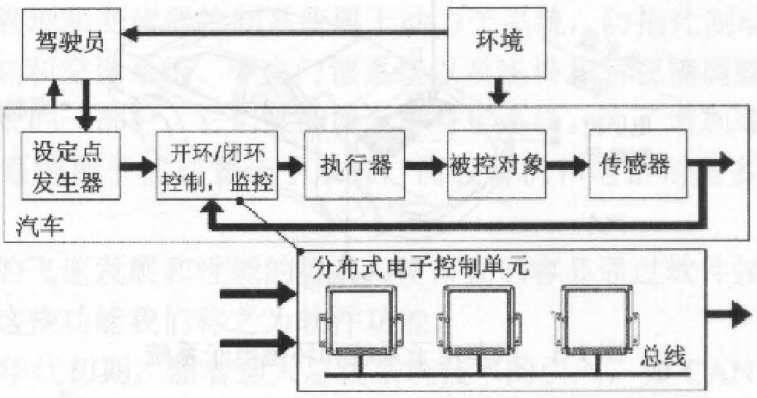


图 1-2　控制及监控系统结构方框图

如图 1-2 所示，有可能在多个组成部件间存在着信号流向，比如在驾驶员、汽车及环境之间。在这个图中，驾驶员作为一个占位符来表示所有的汽车用户，比如驾驶员和乘客。

环境系统包括附近的其他车辆以及相关的电子系统。这些相关的电子系统包括在汽车维修店连接到车载电子控制系统上的诊断测量设备等工具（图 1-1）。

新技术能更好地帮助驾驶员和汽车、驾驶员和环境以及环境和汽车之间交换信息，从而促进更多创新的功能。举个简单的例子，我们能超越车身通过无线通信系统建立网络连接，这种无线方式的网络使一种崭新类型的系统可能得以实现，比如驾驶员辅助支持系统。在过去的几年中，通过车-环境网络，我们得以引入很多基于多媒体系统领域的新功能。例如汽车动态导航功能，它能根据具体的环境信息，比如交通堵塞报告，从而估算出理想的行车路径。

其他一些新的革新也为汽车用户界面领域带来更多的便捷，也就是在驾驶员或者乘客与汽车之间建立互动。在今天，操作系统和显示系统都能基于声控方式得以实现。

正如前面所叙述的，在本书中，我们讨论的网络已经不仅仅局限于汽车电子功能，它超出汽车本身的界限，并包含了驾驶员和汽车互动的电子功能和电子系统。这也能帮助我们更好地理解以下所引入和讨论的术语：车载和非车载、在线和离线。

1.1.2 汽车电子系统和环境

在以下的章节中，我们把汽车中电子系统中的通信称为车载通信（Onboard Communication）。相应地，车载电子系统和环境系统间的通信我们称之为非车载通信（Offboard Communication）。相类似地，当提及汽车电子系统界面的时候我们也区别车载界面与非车载界面。通过图 1-3 我们可以得到一个概括的了解。

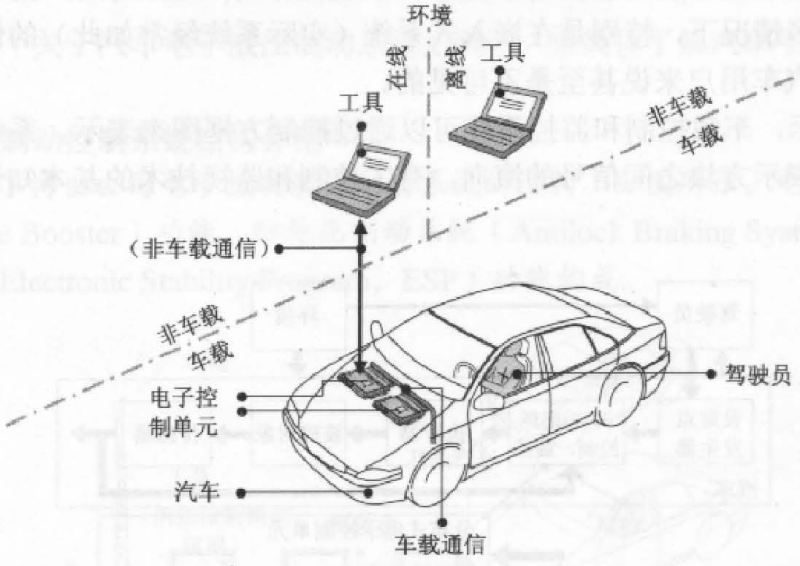


图 1-3 汽车电子系统和环境电子系统

对于由汽车中电子系统执行的功能和由外界环境所处理的功能，我们也区分车载功能和非车载功能。我们可以把功能分割成多个子功能，这些子功能也可以分别由车载电子系统和非车载电子系统来分别实现。

另一个区别是，非车载系统执行某个功能的时刻与车载系统执行某个功能的时刻之间的关系。在这种情况下，我们可以做出同步执行（也称为在线执行）和异步执行（也称为离线执行）的区分。

在有关汽车功能的检测和诊断系统中，车载/非车载和在线/离线的区别已经应用了很多年，很多读者对于车载和非车载诊断概念都不会感到陌生。汽车电子系统的设计方法和工具也以类似的方式进行分类。

1.2　汽车电子系统概述

汽车中通常包含不止一个电子控制元件，以对不同的子系统进行控制和监测。在早期的汽车电子技术应用中，这些电子控制元件的运行在很大程度上是独立的。由于各个电子控制单元中没有相互联系，把与某个汽车子系统（比如传动系、底盘、车身和多媒体子系统等）相关的功能，指定由某一个电子控制单元来负责控制和监视是比较容易的（图 1-4）。

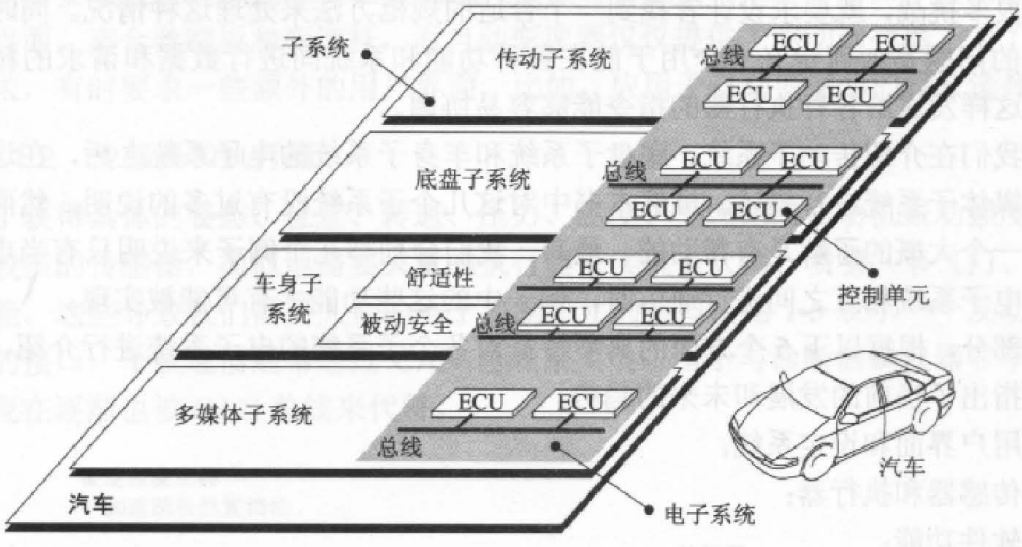


图 1-4　汽车子系统与电子控制单元

因此，发动机管理和变速器控制系统属于动力子系统，防抱死制动系统（ABS）属于底盘子系统，而加热和空调系统、中央门锁系统以及座椅和后视镜调整装置是作为舒适性和提供方便性子系统的一部分，它们又同属于车身子系统。为了增加事故中用户安全性的安全气囊和安全带系统属于被动安全子系统，而收音机和电话则是多媒体子系统的一个部分。

由于硬件技术的飞速发展和性能的提高，使得我们容易通过软件技术来实现许多日益强大的汽车功能，这些功能我们称之为软件功能。

在 20 世纪 90 年代初期，随着强大总线系统技术的引入，如 CAN 总线，开创了电子控制元件技术的第二个阶段[2]。现在电子系统网络化已经成为可能，在此基础上能够实现一些新的更高水平的软件功能，同时其成本大大降低。例如，并不需要昂贵的布线结构，现在多个系统就能从各个传感器接收信号。

只要更高层次级的软件功能能够影响一个子系统中的任何功能，这种实现方式也就是我们所说的集成的发动机子系统、集成的底盘子系统、集成的车身子系统或者集成的安全管理子系统。在分布式网络系统中，通常不再把软件功能指定由某个单独的电子控

制单元来实现。因而，软件功能被划分为多个子功能，并由几个不同的电子控制单元来实现。

只要更高层级的软件功能能够影响多个子系统中的功能，这种软件功能就不再属于某个特定的子系统。就如前面例子中的讨论，驱动力控制系统（TCS）是由一系列功能组成，这些功能影响着发动机子系统和底盘子系统。很多其他的驾驶员辅助支持系统，比如测距自适应巡航控制系统同样也属于这种类别。在舒适性子系统和被动安全子系统中也存在跨越子系统边界的功能。包含门锁子系统和防盗子系统的汽车进入系统就是这类功能中的一种。从功能性的观点看，这些例子强调在各个子系统之间联系的流畅和屏蔽性。

在大多数情况下，多个电子控制单元同时存取和使用多个传感器信号是没有困难的。但是，只要有多个电子控制单元或者多个软件功能同时竞争使用一个执行器，这就给设计者提出了很多挑战，就要求设计者找到一个合适的规范方法来处理这种情况。同时另一个需要面对的问题是如何定义一个用于能在各种功能和系统间进行数据和请求的精确交换的界面，这样发送给各种执行器的指令能够容易协调。

下面我们在介绍传动子系统、底盘子系统和车身子系统的电子系统之后，在这部分还同时对多媒体子系统进行描述。虽然本书中对这几个子系统没有过多的说明，然而对于这些系统有一个大概的了解是有帮助的。最后，我们会列举几个例子来说明只有当电子系统网络化和电子系统相互之间进行通信时，例子中的这些功能才有可能被实现。

在这部分，根据以下 5 个方面的典型特征对各个子系统的电子系统进行介绍，同时在讨论中也指出了当前的发展和未来的趋势。

（1）用户界面和设定系统；

（2）传感器和执行器；

（3）软件功能；

（4）安装空间；

（5）型号和可测量性。

在很多情况下，功能的技术实现必须考虑到一系列法律方面的规定。例如，任何传动系的开发，特别是发动机电子控制单元软件功能的开发，通常都要考虑到相关燃料的节约措施和废气排放标准的规定。相对应地，底盘和车身系统功能必须考虑到安全性和舒适性方面的要求。

本节对汽车电子系统和它们的功能只做简单概述。有关电子系统的功能和系统开发方面的详细内容我们会在以后的章节中通过例子来进一步介绍。如果要对这些内容做更深入的了解，请参考文献[4]。

1.2.1 驱动部分电子系统

一辆汽车的驱动部分包含以下组成部分：

（1）驱动组，比如发动机、电动机、混合动力驱动或者燃料箱；

（2）离合器和手动变速器，或自动变速器；

（3）分动器，前桥或后桥驱动；

（4）传动轴；

（5）发动机辅助系统，例如起动机和交流发电机。

传动系的电子系统包含：

（1）发动机电子控制单元；

（2）变速器电子控制单元。

发动机、变速器和辅助单元实现的许多控制和监测功能通常把驾驶员的要求和一系列的传感器信号作为输入变量，并用来控制驱动部分的执行器。

1.2.1.1　用户界面和设定点发生器

相比较而言，电子动力传动系控制功能有较少的用户界面。除了起动和关闭发动机，驾驶员只需要通过改变加速踏板位置就能直接发送自己的请求。手动操作提供了两个额外的用户界面，离合器踏板和变速杆，而自动变速器仅仅提供了驱动选择器。为了满足一些特定需求，有时要求一些额外的用户界面，比如，应用于自动变速器的模式选择开关。

1.2.1.2　传感器和执行器

为了获得具体的姿态、位置、转速、压力、温度、空燃比或发动机振动强度，我们需要相对较多的传感器，类似地需要大量的执行器来实现点火器、喷嘴、节气门、离合器或阀门功能。这些导致我们需要大量的电子控制单元的接口。图 1-5 表示一个发动机电子控制单元的接口。车载通信通常通过 CAN 总线来实现。用于与诊断测试仪器非车载通信的 K 总线现在逐渐也被 CAN 总线来代替。

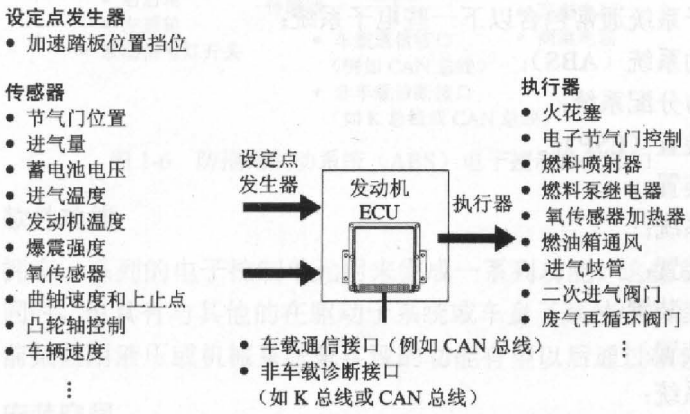


图 1-5　汽油发动机电子控制单元接口[6]

1.2.1.3　软件功能

一个发动机电子控制单元拥有大量的软件功能，目标是开发强大的软件功能，一方面使得这些软件功能在子系统内部协同工作，同时具有与其他子系统中（底盘或车身子系统）软件功能的接口，例如具有与牵引控制系统或空调系统的接口。

很多软件功能具有大量的参数，比如特征值、特征曲线和特征图。开发人员通过改变相应的参数，使软件功能能与相应的发动机、变速器或汽车型号以及不同的工作点相适应。

1.2.1.4 安装空间

由于电子控制单元与传感器、执行器具有大量的接口，而传感器和执行器的大部分又位于发动机或变速器上，因此这些电子控制单元通常安装得非常靠近被控对象，电子控制单元的工作条件也往往比较苛刻。在很多情况下，这些电子控制单元长期暴露在高温、潮湿和振动的环境中。

1.2.1.5 型号系列

一般我们不能满足任何客户的个性化的要求。然而，汽车消费者通常能够从多种发动机和变速器型号中做出选择。

基于以上原因，驱动部分往往具有几个能够处理很多软件功能的强大的电子控制单元。其中必须要有的是发动机电子控制单元，在采用自动变速器的汽车中，还需有额外的变速器电子控制单元[4,6,7,8]。

1.2.2 底盘电子系统

底盘包含以下几个组成部分：

（1）车轴和车轮；

（2）制动装置；

（3）悬架系统和减振器；

（4）转向系统。

因此，底盘电子系统通常包含以下一些电子系统：

（1）防抱死制动系统（ABS）；

（2）电子制动力分配系统；

（3）电子稳定装置（ESP）；

（4）驻车制动装置；

（5）胎压监测系统；

（6）气动悬架装置；

（7）抗侧倾稳定装置；

（8）助力转向装置；

（9）主动转向系统；

（10）电子液压制动器或电子机械制动器；

（11）线控制动和线控转向系统。

对于驾驶员而言，制动系统的功能故障意味着当他踩制动踏板时将会没有响应或没有踩制动踏板就制动了。如果转向功能出现故障，那么当驾驶员转动转向盘的时候也会发现没有响应，或者说不转动转向盘汽车却自己转向。当以上任意一种功能失灵，都会让驾驶员完全丧失对汽车的控制。换言之，当这两个系统出现故障，意外事故和人员伤亡的风险将是相当高的。因此，对于这些系统的安全性要求是非常严格的。为了达到与汽车安全性紧密相关的子系统的要求，设计人员必须遵循严格的设计原则，比如增加监视和安全措施，

想方设法减少与其他系统接口的数量，并使用模块化技术。事实上整个现代汽车电子系统的设计在相当大程度上受这些与安全性紧密相关的子系统的影响。

1.2.2.1　用户界面和设定点发生器

如同在驱动部分的电子系统功能，底盘部分的电子系统具有较少的用户界面。驾驶员通过制动踏板、转向盘和驻车制动这三个用户界面来产生设定信号。驾驶员可以通过组合仪表上的指示灯来确认驻车制动的动作。有些底盘系统，比如气动悬架系统，通过额外的用户界面（控制单元）来使之工作或不工作。

1.2.2.2　传感器和执行器

在防抱死制动系统（ABS）电子控制单元中，车轮转速是其中的一个主要输入变量。车轮制动器中包含了执行器。电子稳定系统控制单元则使用另外的输入信号，例如转向盘角度传感器信号和偏航角传感器信号[1]。与驱动部分电子系统或发动机电子控制单元相比，这里使用的传感器和执行器会相对少一些。图 1-6 示意了防抱死制动系统（ABS）电子控制单元的接口。

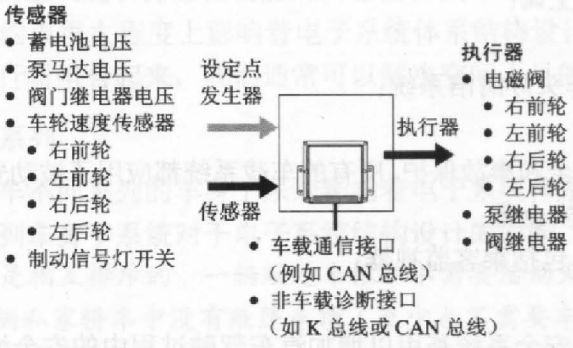


图 1-6　防抱死制动系统（ABS）电子控制单元接口

1.2.2.3　软件功能

底盘系统拥有一系列的电子控制单元用来完成一系列功能。这些功能在底盘子系统内部协调工作的同时，也具有与其他的在驱动子系统或车身子系统中的软件功能相连接的接口。因此，当前某些用液压或机械系统来实现的功能有望以后通过软件的方式来实现。

1.2.2.4　安装空间

由于高度的安全要求，也由于底盘传感器和执行器界面所需空间分配比较大，所以底盘组的电子控制单元数目要大于驱动部分所含单元数目。底盘系统包括一个 ABS ECU，一个 ABS/TCS ECU 或者一个 ESP ECU，以及用于转向盘、悬架和阻尼减振或胎压监控器的可选的 ECU。和驱动部分中的 ECU 一样，这些 ECU 也暴露于类似的苛刻条件下[1]。

1.2.2.5　型号系列

根据市场上对于汽车种类型号需求的不同，底盘部分有着不同的装配标准。另外，也

会提供一些专用功能用于备选。在很多市场上，消费者可以挑选以及组合各种不同的备选功能。

1.2.3 车身电子系统

车身电子系统通常划分为与乘客安全性相关以及与舒适性/便利性相关两部分。与舒适性/便利性相关部分包括以下的组成内容：

（1）汽车门锁系统，包括中央锁定系统、无线钥匙和防盗系统以及操纵系统；
（2）电动窗和活动车顶；
（3）滑动天窗；
（4）活动顶篷；
（5）雨刷和雨量传感器；
（6）车镜调节器、调光器和加热器；
（7）座位调节器和加热器；
（8）转向盘伸缩调节器；
（9）车内暖风装置和空调；
（10）车内照明；
（11）车头灯控制和车头灯清洁系统；
（12）停车援助功能。

为了增加汽车用户安全和事故保护，所有的车载系统都应用了被动安全系统，其中包括：

（1）约束系统，比如安全带；
（2）安全气囊 ECU，包括乘客监视器；
（3）卷收器。

相对而言，车载主动安全系统是用以增加汽车驾驶过程中的安全性，它致力于汽车驾驶的临界控制从而避免事故的发生。防抱死制动系统和电子稳定装置就属于这类系统。

1.2.3.1 用户界面和设定点发生器

用户界面众多是舒适性/便利性子系统的一个特点。驾驶员和乘客可以使用大量设定点发生器，比如开关、按钮、滑动片或旋钮来调整他们座椅取暖器、高度和转向盘的角度等。换言之，用户可以通过控制设定点发生器而频繁地向系统发出要求（并由此一定程度上能意识到与系统的交互），用户也可以直接感受到系统对用户要求的响应。因此，用户一般都十分了解这些方便的功能。

相反，被动安全系统基本上没有任何用户界面。所以只有当用户非常重视汽车安全，才会注意到这些功能的存在。

1.2.3.2 传感器和执行器

各种车身子系统功能的输入包含一些指定的标称值和一系列的传感器信号。在很多情况下，执行器是通过电子驱动器来实现的。

1.2.3.3　软件功能

车身系统中存在着大量独立的软件功能。在典型的自主应用模块中，电子控制单元应用了计算能力有限和输入/输出接口相对较少的微控制器。通过这种方式，在独立的 ECU 中就能执行独立的软件功能。在这种应用方式下，通常需要大量的 ECU。

通过把多个电子控制单元的网络化可以更好地完成超越单个 ECU 和子系统范围的软件功能的实现，中央门锁系统就是一个典型的例子。

相对于传动系统或底盘子系统中的软件功能，车身电子软件功能中所用到的标定参数就少了很多。

1.2.3.4　安装空间

汽车上分布了各种各样的 ECU，很多情况下，它们能在其安装位置直接执行它们的功能。车门、车外镜、车顶、车尾和车端部分以及车内部分的传感器与执行器都必须与他们各自的 ECU 相连接。通常情况下，给予 ECU 和配线布置的安装空间是相当有限的，比如车门内或车座。在有些型号的汽车中，几个传感器、执行器和 ECU 同时争用同一个安装空间。例如，用于安全气囊和舒适性功能的传感器和执行器都必须安置在车门、座位或者车顶。很显然，空间约束很大程度上影响着电子系统体系结构设计。通过把有限功能的 ECU 和智能传感器及执行器组合起来，我们通常可以解决空间不足的问题。

1.2.3.5　型号系列

同一种型号汽车不同系列的车身子系统影响着电子系统体系结构的设计。

举例：不同系列车身子系统对于电子系统结构设计的影响

有些设备选项是相互排斥的。一辆敞篷车根本不需要活动天窗（所以也不需要此种功能的 ECU），而一辆私家轿车中没有敞篷车顶（至少也不需要车棚上/下功能的 ECU）。类似的情形也存在于旅行汽车和供两人坐的小汽车之间。这也是为什么很多 ECU 只提供有限功能的一个原因（模块化）。

除了不同汽车型号的要求，系统的不同配置规模的要求也必须满足。用户可以从一系列不同的车身子系统功能中做出自主选择。

1.2.4　多媒体系统

多媒体子系统由以下部分组成：

- 调谐器和天线
- CD 自动换片器
- 音响和音频系统
- 视频系统
- 导航系统
- 电话
- 声控系统
- Internet 接入系统

这些电子系统的其他额外的功能只有通过与其他的子系统的电子系统的网络化来实现，比如，通过声控或可视方面获得的舒适功能。

1.2.5 分布式网络电子系统

ECU 网络化使得综合的（跨单个电子控制单元或子系统）软件功能的实现变得更加便利。

举例：自适应巡航控制（ACC）系统

自适应巡航控制系统是在传统的巡航控制基础之上的进一步发展。主车配备了一个前视传感器（例如雷达）来测量与前方车辆的车距、前方车辆的车速以及相关位置。自适应巡航电子控制单元就是用这个传感器输入来进行计算并使主车和前方车辆保持安全距离。电子控制单元经过计算后，通过增加或者减少主车的纵向动力来达到和保持这个安全车距。其具体工作方式是，通过干预发动机 ECU 来影响发动机转速，干预变速器 ECU 来影响传动，以及干预 ESP ECU 来影响制动扭矩。所以 ACC 包含的功能可以影响传动子系统和底盘子系统。

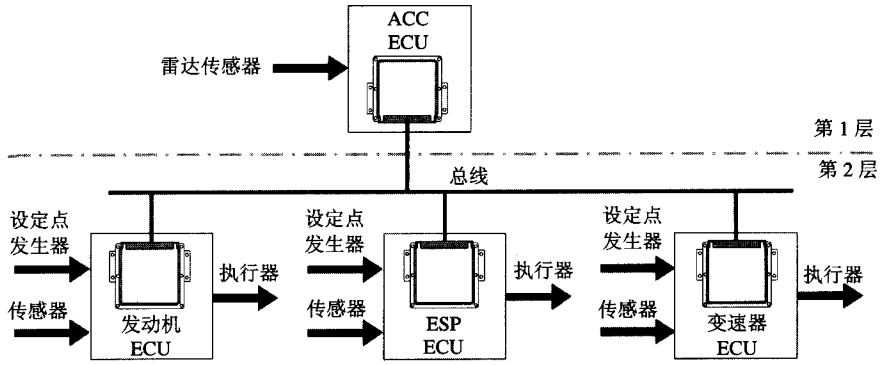


图 1-7 ACC 电子控制单元和 ACC 系统

举例：提示、警告和故障信息显示系统

显示在仪表上的提示、警告和故障信息都是由各种 ECU 发送的数据。例如，仪表盘单元软件功能的任务不只是显示来自于车载 ECU 的提示、警告和故障信息，还对不同的信息指定不同的优先级排列。在提示、警告和故障信息显示前，系统会根据一定的策略对来自所有 ECU 的信息进行评估（图 1-8）。

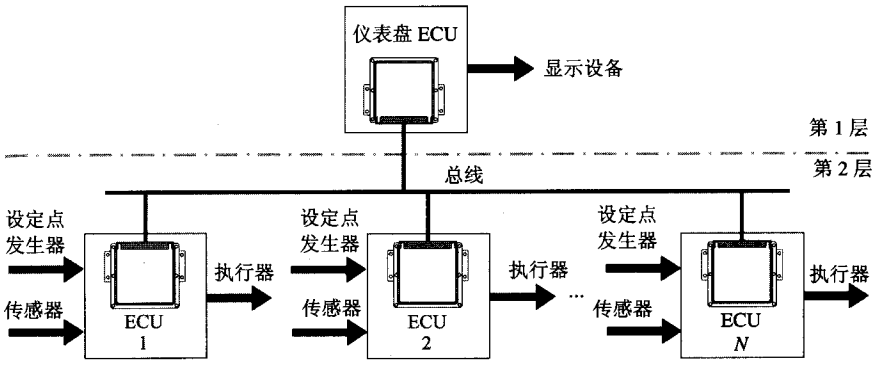


图 1-8 提示、警告和故障信息显示系统

1.2.6　小结和展望

微控制器性能的持续改善很大程度上影响着电子系统的技术实现，事实上，如今多个电子系统之间大多是网络化的。

在过去的几十年里，不单单每辆汽车中的 ECU 数量在增加，每个 ECU 所包含的功能数量也以稳定速率在增加（图 1-9）。

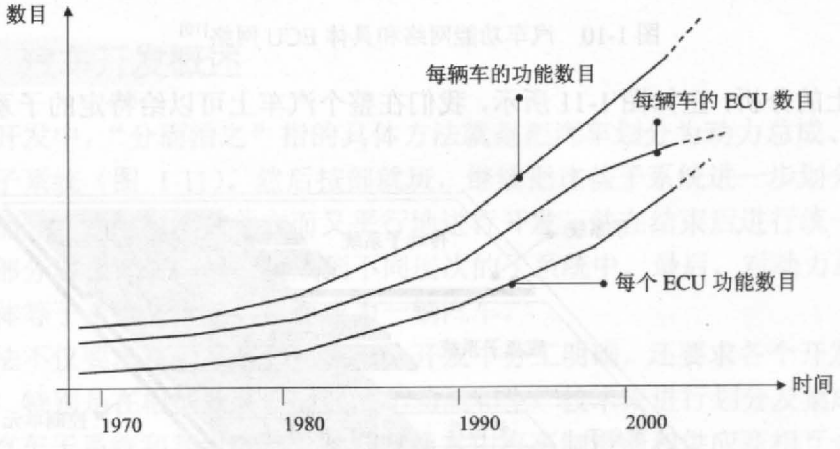


图 1-9　一辆汽车中的功能和 ECU 数目

现在我们可以预见将来每辆汽车的功能数目会持续增长。就如之前所说，很多新的功能只有通过汽车和周围环境之间的网络化才能得以实现。如今由机械或液压系统实现的一些传统功能，在未来至少部分可由软件功能来实现。

随着豪华汽车的推进，我们也可以预见各种类型汽车标准配置中附加功能的增加及深入发展的趋势将会持续。然而，成本增长和小型汽车空间的局限性同时要求减少 ECU 数量，或者至少 ECU 的数量将不会增加。例如，如果可用安装空间有限制，那么目前由多个 ECU 执行的各种软件功能在将来可能只需要一个 ECU 来实现。这也是单个 ECU 实现软件功能数量将持续增长的另一个原因。

1.3　逻辑体系结构概述

当电子控制组件刚开始在汽车中应用时，典型 ECU 的使用示意图如图 1-4 所示。由于各个独立系统之间都是独立工作，功能分配也相当清晰，所以在当时这种模型是非常正确的。在那种情况下，ECU 就相当于具体功能。

1.3.1　汽车中的 ECU 网络和功能网络

前面章节中我们已经举例介绍了自适应巡航控制系统，然而，要强调的是，与基于独立的 ECU 相比，这类网络系统的设计和开发方式有所不同。分布式网络软件系统的开发要求在两个不同层面上来分析和描述系统。分布式网络的功能代表的是系统的抽象层面，

而汽车上各种具体网络 ECU 可以看作是系统的技术实现层面（图 1-10）[10]。

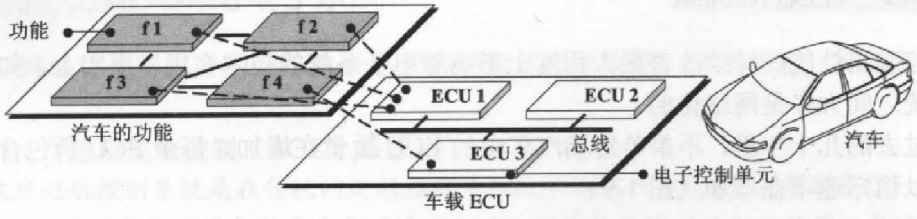


图 1-10 汽车功能网络和具体 ECU 网络[10]

基于以上的分析，正如图 1-11 所示，我们在整个汽车上可以给特定的子系统分配相应的功能。

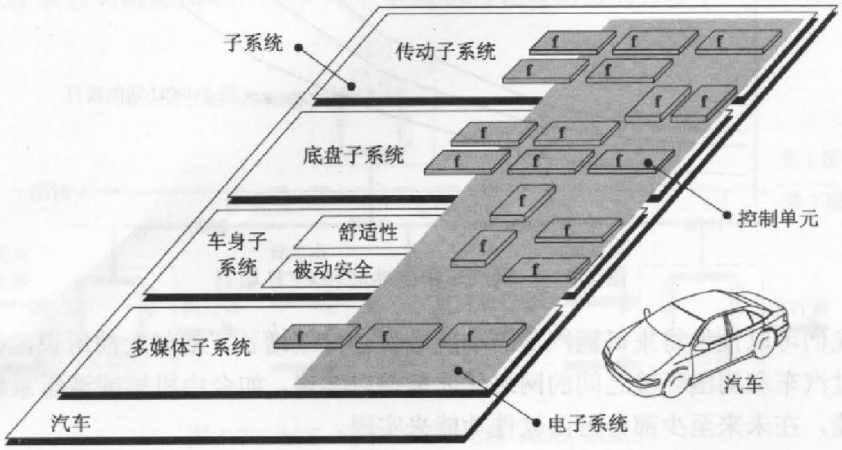


图 1-11 汽车功能在子系统上的分配

1.3.2 开环/闭环控制和监视系统的逻辑体系结构

在汽车各个组成部分中以及驾驶员和环境之间，抽象观念和具体观念的区别随处可见。在以下讨论中，抽象层面被称为系统逻辑体系结构，而具体实现层面被称为系统技术体系结构。为了区别清楚，书中以一个灰色背景来表现系统逻辑体系结构，以一个白色背景来表现系统技术体系结构。图 1-12 示意了开环/闭环控制和监视系统逻辑体系结构，图 1-2 示意了系统技术体系结构。

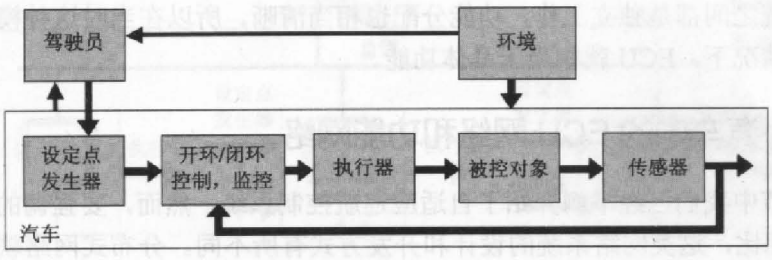


图 1-12 开环/闭环控制和监视系统逻辑体系结构

1.4　汽车的开发过程

随着汽车功能的增加、网络的应用以及汽车可靠性、舒适性、安全性以及型号和可扩展性要求的不断提高，就要求有一个详细制定的开发步骤，否则将难以处理好如此复杂而众多的问题。目前可用于而且已经长期应用于汽车工业的一种方法可以很好地处理这些复杂的问题，称之为：分而治之。

1.4.1　汽车开发概述

在汽车开发中，“分而治之”指的具体方法就是把汽车划分为动力总成、底盘、车身和多媒体等子系统（图 1-11）。然后按部就班，继续把这些子系统进一步划分为次级子系统和各组成部分。各组成部分独立而又平行地进行开发，并在结束后进行统一测试。随后对这些组成部分进行评估，然后集成到不同层次的子系统中。最后，对动力总成、底盘、车身和多媒体等子系统进行集成从而成为一辆汽车。

这个方法不仅要求在子系统和组成部分开发中分工明确，还要求各个开发小组之间都要相互协作，特别是在根据安装空间、汽车功能和生产技术来进行划分及集成系统时。

另外，汽车子系统和其组成部分的开发通常由汽车制造商和供应商相互合作完成。正因如此，一个清晰准确的任务分配也是汽车成功开发的一个基本要求。

不同汽车的同时开发及汽车的型号类型带来另一个要注意的问题。对汽车制造商和他们的供应商来说，这就意味着要有一个日常程序，用来工作于系统的所有层次上的多个项目。

不同工程学科和不同公司之间要进行协作，就需要他们要在中心原则问题上共同熟悉，在解决问题过程中相互共享与理解，对于将用于全程的解决方案的效果和可能存在的问题有一个合适的评价。并且对于一个给定方案，必须详细定义相关责任和义务。在一个开发方案中，技术层面上的机电一体化或组织层面上的系统工程方法也可以作为例证或是协调方式。

汽车制造商和供应商间的相互合作中，必须确定商业模式的各个方面，以及产品责任或专利权等法律问题。不过本书只限制于技术和组织方面的讨论。

1.4.2　电子系统开发概述

汽车电子系统的开发过程步骤与前文提到的汽车开发类似。因此，首先要做的也是把汽车电子系统划分为子系统，比如 ECU（硬件和软件）、设定点发生器、传感器和执行器（图 1-2），这些都是基于分工原则来实行的。接着对各子系统进行测试和确认，进而一步步集成为一个完整的电子系统（图 1-13）。在这里，划分和集成同样需要超过子系统边界的合作。

电子系统开发应该遵循可行的方法精心规划和准备，比如能力成熟度模型集成 [13]，软件过程改进和能力测定（SPICE）[14]，或是 V 模型[15]。

此外，汽车电子系统和组成部分也应支持自动化标准，比如 OSEK[16]和 ASAM[17]。OSEK 名称来自于德文“车内电子设备的开放系统的接口”。ASAM 则指自动化和测量系

统标准化组织。

另外，在定义一个开发方案时也应考虑到验证测试程序，比如模拟仿真或快速原型制造技术。

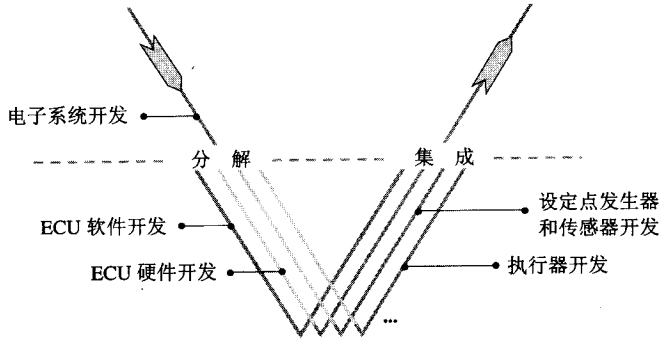


图 1-13 电子系统开发概述

对于电子系统开发方法，一般推荐做到以上提到的几点要求，当然也要求在汽车开发和处理电子系统开发时设计员之间大量的互动与交流。这些交流互动也将会在软件开发中继续延续。

1.4.2.1 由硬件到软件的趋势

电子系统开发总体上体现了由基于硬件的解决方案转到基于软件的解决方法。

软件解决方法非常适合来实现电子系统的功能。举个简单的例子，基于软件的开环/闭环控制功能和监控功能的执行有着非常高的自由度，比方说在适应性、学习算法、安全性、诊断学理论以及线性化设计方面。换言之，选择这种方法来执行实现可以很大程度上忽略安装空间以及实际制造限制。

由于以上这些原因，基于软件来实现执行汽车功能给汽车制造商和供应商带来了巨大而截然不同的潜在竞争力，在其他行业中也是如此。

因此，本书的着眼点在于目前已经广泛应用于软件功能开发以及这些功能在汽车电子系统中和其他部件相集成的方法。

其中特别强调的是应用于汽车 ECU 软件开发的需求描述和应用约束的描述。这些与应用于其他行业中 ECU 软件开发的要求有非常显著的不同。为使读者详细了解这些不同，本书将会对这些具体要求进行对比介绍。

例如，ECU 软件功能规格说明和它们的实际设计实现有着严格的区别。一个给定软件功能的规格说明指的是它在开发及在早期阶段确认它在实际汽车上的功能。而设计和实现指的是使软件功能与特定的目标 ECU（由技术上实现）相适应，然后把结果与给定规格进行对比验证。汽车产品和汽车服务的额外要求一般也由汽车 ECU 使用的软件来实现。

1.4.2.2 成本

低成本而高产量差异所带来的巨大压力，是一般汽车工业的普遍特色，因此在汽车行业中，相应成本通常指的就是单位成本。实现降低 ECU 单位成本则转化为有限的内存以

及有限的运算能力。因而，软件开发员就想方设法尽可能使之最优化，比如尽可能在可行的条件下使用整数型算法来实现功能执行。

1.4.2.3 较长的产品生命周期

就目前情况而言，我们以如下所列来评价汽车的生命周期：

（1）3年的开发阶段；

（2）约7年的产品生产阶段；

（3）一直到15年左右的运转和保养阶段。

就以上所说，汽车的完整产品生命周期是25年左右（图1-14）。

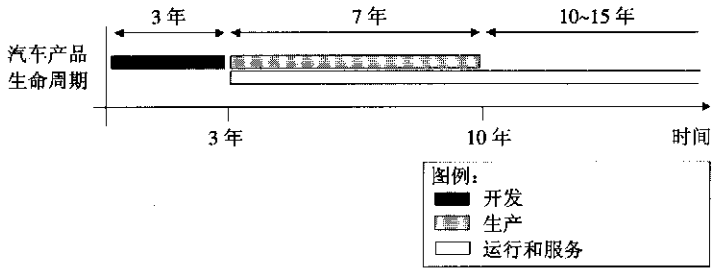


图1-14 汽车产品生命周期

然而对于其中的电子组成部分而言，这个过程会由于硬件技术的持续进步而显著缩短。这样产生的一个结果就是，电子备用配件长期的市场供应会面对相当大的风险与挑战。很明显，在汽车开发阶段必须慎重考虑诸如此类的风险挑战。

电子硬件寿命周期的必然缩短也对软件结构体系带来冲击和影响。软件结构体系标准化趋势就是受这种冲击影响的一个例子。另外，为了确保新一代硬件的软件功能接口简化，硬件也朝着规格独立化趋势发展。

由于工作在ECU中的软件不断更新升级，所以目前看来，汽车中ECU软件生命周期要比ECU硬件的生命周期来得短些。闪存技术的发展，使得车载ECU可以很容易地通过网络连接进行程序改编，这也促使了这个领域中高成本效益软件的不断更新。这种程序改编是通过汽车中央外诊断接口进行的，并不需要成本昂贵的汽车ECU移动和交换。因此，在开发过程中必须考虑如何尽量延长汽车寿命周期。

1.4.2.4 不断提高的安全性要求

与其他行业的安全性要求相比，汽车功能的安全性要求要严格很多，比如要严格遵循系统制造工程学科或电子通信学科的要求。这是为了保证一旦发生意外事故，能最大程度确保驾驶员的安全。因此，各个功能都按照高度完整的安全等级进行分类。而其他机械制造行业中，由于可以对工人进行适当的约束限制来降低此类意外事故和人员伤亡的可能性，所以相对而言不用过多考虑这方面的问题。

很多标准中都定义了基本安全性规定，例如DIN 19250标准[18]，IEC 61508[19]以及ECE Directives（欧盟法令）[20, 21]。一辆汽车要获得注册许可和通行证，必须经过一系列安全性检验。

近些年，由于汽车电子系统在汽车安全相关工作中的作用越来越突出，因此其本身工作的安全可靠性变得尤为重要。其功能的范围很广，包括从提供具体分析（例如速度显示器）到给出评估信息（例如黑冰预警）；或从行进动作推荐（导航系统）到具体执行（加速或制动干预）；甚至能纠正驾驶员的错误驾驶动作，如转向盘主动干预（在装有主动朝向干预或 AFS 系统的汽车中）[22]。

正因如此，运行的安全性分析很大程度上影响着功能开发，进一步说就是软件开发。迫切的可靠性要求使故障检测、故障处理以及冗余系统的开发成为所有环节中最为重要的部分。事实上，安全性要求也增强了汽车系统分布式和网络化的趋势。

这些考虑事项和限制也迫使我们对开发步骤和开发工具有了特殊的要求。比如 OSEK 操作系统就是一个对开发工具和标准化软件组成有特殊要求的一个典型例证。

1.4.3 电子系统和软件开发的核心过程

汽车、电子系统和软件开发间存在着千丝万缕的相互联系，使得我们必须要有一个综合完整的开发过程。这个过程包含开发的各个步骤，从用户需求分析到最终的电子系统的验收测试。

本书着重介绍基于 V 模型的程序化的方式来进行电子系统和软件的开发。V 模型分别从整体层面和组件层面把质量检查和测试结合起来。这就是 V 模型在汽车行业中得到广泛应用的原因。

开发过程参考模式可以用字母“V”来表示。一个合适的 V 模型为开发过程阶段以及系统与软件开发间界面提供正确的表述，具体到汽车开发的过程也是如此。图 1-15 大体显示了我们所说的核心流程，这个核心流程将在本书第 4 章中再进行详细介绍。支持这个核心流程的理论方法和所用工具将在第 5 章中作介绍。

核心流程包括一系列不同的开发步骤：

1. 用户需求分析和系统逻辑体系结构的确定

这个步骤的目的是基于与项目相关的用户的需求，对系统逻辑体系结构进行详细定义说明。系统逻辑体系结构包括了对功能网络、功能界面和整个汽车（某种情况下也可以是单个子系统）中各项功能间通信的详细定义说明。这个过程并不对具体的技术实现方面做出任何决定。

2. 系统逻辑体系结构的分析和系统技术体系结构的确定

系统逻辑体系结构是确定系统技术体系结构的基础。各种技术实现方案的分析是建立在统一的系统逻辑体系结构的基础之上，并得到一系列多方参与的工程学科的理论与方法的支持。系统技术体系结构包括了所有通过软件实现的功能以及子功能的定义说明。这个定义也称之为软件需求的确定。

3. 软件需求的分析和软件体系结构的确定

对软件的需求进行分析，并确定软件的体系结构，具体就是确定软件系统的边界和接口，确定软件的组件以及软件层和软件的工作模式和状态。

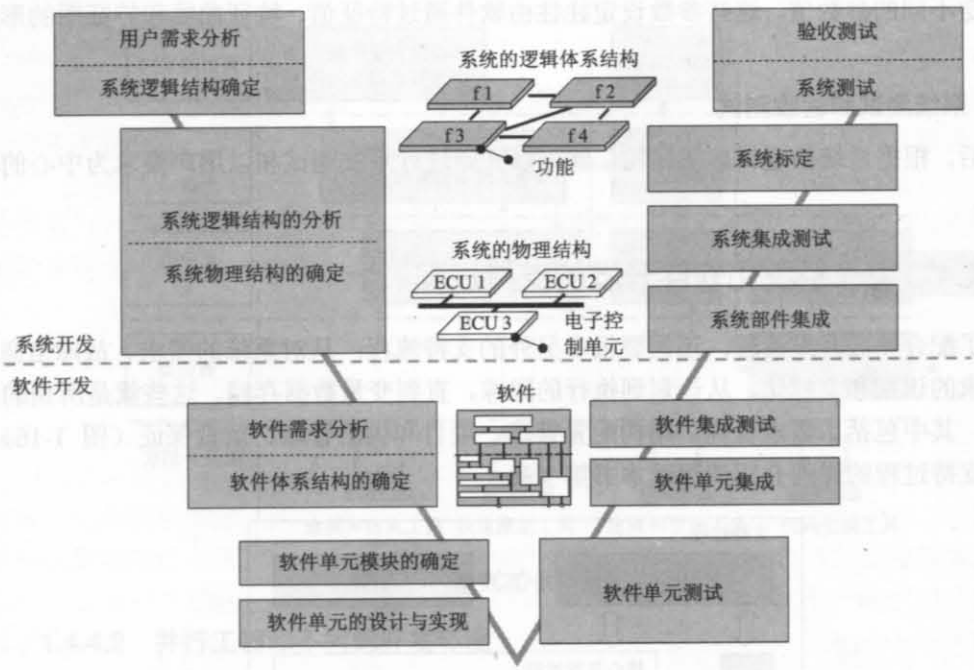


图 1-15　电子系统和软件开发核心流程

4．软件组件的确定

这个步骤详细定义软件的组件。在这一步骤中我们开始假定处于理想状态环境下。这意味着整个步骤中可以完全不用理会任何执行细节，比如整型算法执行。

5．软件组件的设计、实现和测试

在设计阶段，上一步骤假定的理想状态将被重新审视。在此，所有会影响到实际实现的细节都必须进行定义说明。由此得到的设计方案将支配软件组件的实现。在这个步骤的最后，还要对软件组件进行测试。

6．软件组件的集成和软件的集成测试

一旦软件组件开发完成——当然这个过程的进行要遵循分工原则，而且组成部分已经通过了接下来的测试——这时就能开始集成过程了。在各个组成部分整合入一个软件系统中之后，最后进行软件的集成测试。

7．系统组件的集成和系统的综合测试

为了给各个 ECU 提供相应的功能，必须把软件装载到相应的 ECU 硬件上。然后 ECU 要与其他电子系统组件集成起来，比如设定点发生器、传感器和执行器。在接下来的系统综合测试中，对所有系统设备的交互响应进行评估。

8．校准

ECU 软件功能的校准包括了他们的参数确定，常常根据每种不同类型的给定汽车我们

分别确定不同的参数值。这些参数设定往往由软件通过特征值、特征曲线和特征图的形式来提供。

9. 系统测试和验收测试

最后，根据系统的逻辑体系结构，就可以开始进行系统测试和以用户要求为中心的验收测试。

1.4.4 电子系统和软件开发的支持过程

为了配合核心开发流程，还需要很多另外的支持流程：从对系统的需求、故障信息、修改要求的识别和文档化，从计划到执行的跟踪，直到变量数据存档。这些就是所谓的支持过程，其中包括了需求管理、结构配置管理、项目和供给管理、质量保证（图 1-16）。有关于支持过程的详细介绍请阅读本书第 3 章。

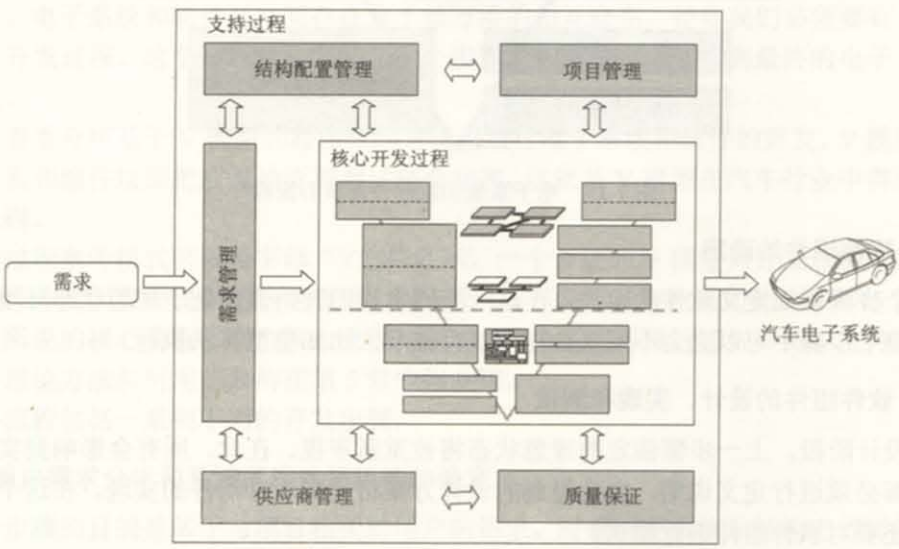


图 1-16 电子系统和软件开发的支持过程概况

为了保证电子系统和软件开发的持续发展，必须管理、支持和集成多个不同的团体和众多的任务。这些包括所有的开发步骤，公司间和公司内消费者/供应商的关系，中间的开发成果，并行开发和各开发步骤间的转换/同步点。与实际商业过程类似，开发过程往往也能以图表形式清晰地表述出来。

1.4.4.1 消费者/供应商关系

图 1-17 给出了一个消费者/供应商关系结构图表化过程[23]。当然高效率的合作团队，意味着方法和工具的高度综合。

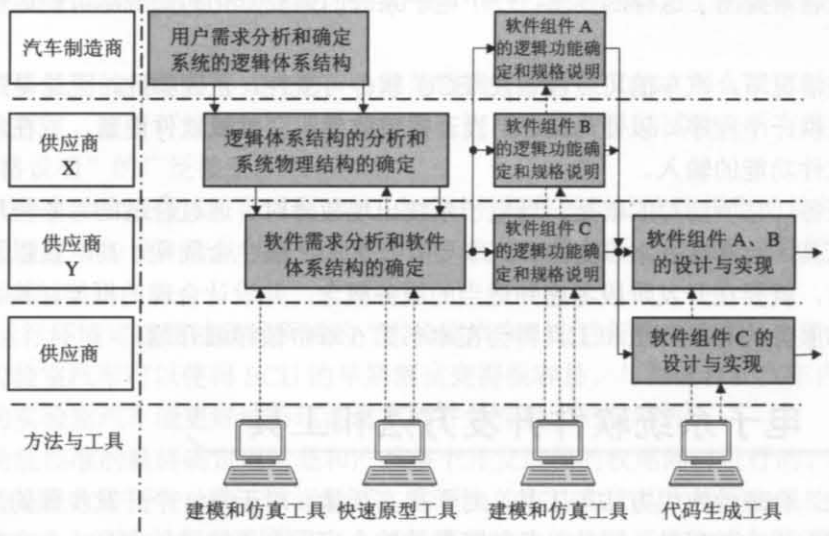


图 1-17　消费者/供应商关系表 [23]

1.4.4.2　并行工程和不同的开发环境

很多情况下，要缩短开发时间，就要求多个开发任务的同时进行，例如并行工程。软件开发中，并行工程意味着对一个软件功能的分析、说明、设计、实现和集成之后进行测试和校准工作，而在这同时也进行其他的软件功能开发工作。另外，还要针对不同的开发环境进行适当调整或理想化整合；也就是在实验室、测试台和在汽车中进行的仿真模拟开发步骤，必须尽可能按照高度标准化来设计，然后对彼此进行同步协调。图 1-18 显示了不同开发环境中列举的并行工程的结构。

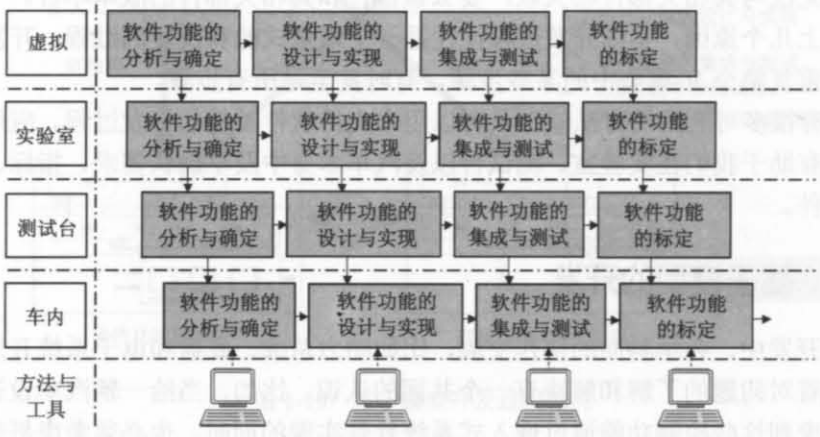


图 1-18　不同开发环境和并行工程

1.4.5　电子系统和软件的生产及服务

通常来说，在生产和服务方面，软件的不同型号/系列的变化相比较硬件容易进行处理

操作，因此通常提出了这样的要求：一个电子系统的需要变化的部分尽可能地通过软件来实现。

在这些情况下，汽车的可变性引发 ECU 软件可变性。正因如此，无论是产品还是服务都必须提供一个程序，以便为 ECU 设计提供软件可变性或软件更新，或在制造过程结束时进行软件功能的输入。

所谓服务，它所面对的就是，当电子系统出现故障时，通过合适的诊断程序，使用相关界面和工具，给修理方案以支持。汽车要有长久的产品生命周期、高产量以及世界范围的市场分布，就要在开发阶段考虑到适当的服务概念，并设计合理的框架结构。

产品和服务的有关方法和工具将会在本书第 6 章中做详细介绍。

1.5 电子系统软件开发方法和工具

事实上，合理地应用方法与工具，对于提高质量，对于每一个开发步骤的风险预防和成本降低都有很大的好处。因此，各种工具的综合应用有着特殊的意义。在本节我们讨论就工具综合应用的方法，和它们对于质量、风险、成本上所带来的影响。

按照 V 模型的论点，暗示了在开发初期，就要完全确定和分析用户需求，并从中可能导出一个足够准确的技术系统体系结构。集成则基于一系列连续的被定义过的步骤。

然而，经验表明，很多情况下这些先决条件并没有得到满足。通常在开始阶段，还不能充分了解用户需求，用户需求往往随着开发工作的进行而随之修正。因此，开始的规划往往只是反映一个大体粗糙的系统概念；其详细定义是逐渐形成的。在系统集成中，相关组成部分的延迟会造成整个集成过程的延迟，以及之后所有步骤的延迟。如果软件开发任务是由不同公司来处理，那么对于一个给定部件的集成和测试环节执行往往会受到限制，要么该部件无法与其相关部件相关联，要么该部件和其相关部件的版本不合。

基于以上几个原因，实际开发过程往往是一个增量式的和反复的过程，开发设计人员往往要多次重复整个 V-模型中的某些步骤，有时甚至是所有步骤。

然而，有很多可利用的方法或工具可以用来支持软件功能的开发过程。应用这些方法和工具，会有助于我们在实验室、测试台以及汽车本身中及早确认需求、指标、规范和已经实现的部件。

1.5.1 基于模型的开发

在软件开发中，各学科间的协作交流，比如动力总成、底盘和电子系统开发之间的合作，就意味着对问题的了解和解决有一个共同的认识。比如，当给一辆汽车设计控制功能时，除了考虑到这些控制功能通过嵌入式系统软件实现的同时，也必须考虑舒适性和安全性方面问题。

一个包括所有系统部件的图形化的功能模型通常有助于了解系统的功能。因此，用图形符号表示的基于模型的软件开发方法，比如方块图和有限状态机，正在逐渐取代文本式的软件规范说明。

除了有助于对问题本身和解决方式的了解，软件功能的模型也能给我们提供其他的帮助。

如果规格说明的模型是形式化的，也就是明确且没有歧义的，那么就可以放到电脑上进行仿真执行了。接着可以通过快速原型技术在汽车上进行早期试验。所有这些都能促进“数字化规格说明”的广泛接受。

使用代码自动生成技术，可以在 ECU 上以软件组件的形式实现指定的软件功能模型。为此，必须通过增加设计信息以增强功能模型，这些信息包括必需的非功能性产品特性，比如优化措施。

ECU 运行环境可以通过我们所说的“实验室汽车”来进行仿真模拟。由于处在实验室状态下，实验室汽车可以使得 ECU 的早期测试变得很容易。与测试台和汽车内测试相比，供测试用的实验室汽车能更好地再现测试案例并具有更大的灵活性。

软件功能校准的最终确定通常是和产品整个开发过程的收尾同时进行的。在很多情况下这个过程是在汽车所有功能正常运转后，在适当的方法和工具支持下完成的。

总的来说，如图 1-19 所示[24]，一个基于模型的软件功能开发方式包括了许多定义明确的独立开发步骤。

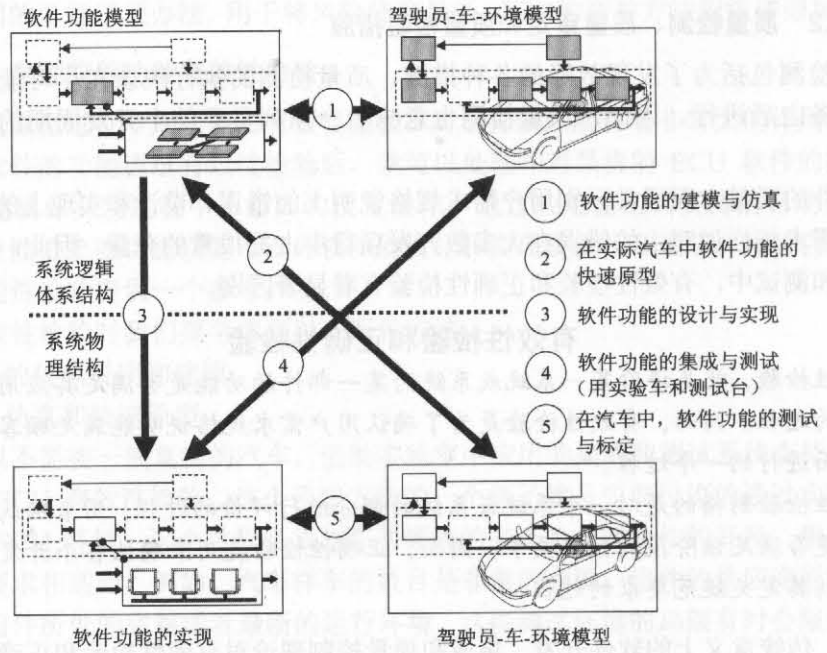


图 1-19　基于模型开发过程概况

一个基于模型的开发方式也同样适合于功能网络和网络 ECU 开发。而这个过程也可以获得额外的自由度，例如：

（1）模型化的、虚拟功能和已实现功能的结合；

（2）模型化的、虚拟的和已实现的技术组件的结合。

1.5.2 综合质量管理

任何软件体系设计方法的目的都是一样的，那就是创造出更高质量的软件。软件质量特性包括了足够的功能覆盖面、可靠性、可用性、效率、适应性和可移植性。

质量管理的内容覆盖了所有那些能保证一个给定产品达到一定要求的方法。当质量保证指引、质量控制及质量测试措施都被制定并遵循时，质量这个概念就“嵌入”到一个产品中了。

1.5.2.1 质量保证指引

质量保证包括了一系列预防措施，比如：

(1) 雇用有着适当受教育程度、工作经验和培训基础的开发人员；

(2) 使用适当、有详细规划的开发流程；

(3) 应用合适的指引、措施和标准来支持这个流程；

(4) 使用适当的工具环境；

(5) 容易出错过程的自动化。

1.5.2.2 质量检测、质量审定和质量检验措施

质量检测包括为了发现故障的各种措施。质量检测的执行应该在尽可能多的开发过程独立任务之后进行。因此，质量检测也意味着一系列贯穿整个开发周期的预定任务的执行。

在软件的质量方面通常区别用户需求规格说明上的错误、设计和实现上的错误。研究表明用户需求规格说明上的错误在大多数开发项目中占据很重的分量。因此，在V模型的质量控制和测试中，有效性检验和正确性检验有着显著区别。

有效性检验和正确性检验

有效性检验，就是评价某一系统或系统的某一部件的功能是否满足其应用目的或满足顾客需求的过程。因而，有效性检验是为了确认用户需求规格说明能满足顾客期望并能被顾客接受而进行的一个过程。

正确性检验则指的是对一个系统或系统的部分进行评价的过程，用来确认给定开发阶段的成果是否满足该阶段的预定要求。因而，正确性检验是用来确认各个开发阶段执行结果切实达到其定义规范要求的过程。

通常，传统意义上的软件开发、集成和质量控制理论对有效性检验和正确性检验并没有明确的界定。与传统工具相比，现代开发工具具有明显优势。应用现代开发工具，即便没有实际现场 ECU 也可以对汽车功能进行有效性检验，因为现代开发工具支持拥有一个实验系统的快速原型技术。

图 1-20 显示了通过应用仿真模拟、快速原型和代码生成工具进行的有效性检验和正确性检验。



图 1-20　有效性检验和正确性检验，包括 ECU 仿真模拟、快速原型和代码生成器

1.5.3　减少开发风险

所谓风险，就是对于可能会给计划方案流程带来严重妨碍的事件。在功能开发中，目前有许多不同的可行干预办法，用于将风险降到最低。其中的两种方法应该详细加以讨论。

1.5.3.1　早期软件功能有效性检验

应用快速原型进行早期功能的有效性检验，对于抑制风险有着非常大的作用，当我们在汽车上对软件的功能通过有效性检验后，就可以处理相对昂贵的 ECU 软件的执行了。这样就能避免软件开发过程中不必要的反复操作。通过应用由工具支持的自动代码生成器，经过有效性检验的功能模型可以为软件的设计和实现提供规范说明。同时它可以为以后的软件正确性检验提供一个参考。

早期有效性检验时我们要考虑到以下两种方法：

（1）通用的规范说明和建模；

（2）模拟仿真和快速原型。

我们可以不需要一辆真实的汽车，借助实验室中应用的集成和测试系统支持，就可以进行早期的 ECU 有效性检验。这个思想方法的一个例子就是先前讨论的通过实验室汽车来模拟仿真 ECU 环境。这个过程必须与那些频繁的多个企业间协作的开发、集成和测试任务的特殊要求相适应。例如，汽车样车的数目是非常的有限。组件的供应商通常不具备他们生产的组件所处的完整或者最新的运行环境。这些测试环境的局限有时会限制了可能的测试步骤。

组件的集成是所有参与的构成部件开发的一个同步设定点。综合测试、系统测试和验收测试只有在所有部件正式完成并集成后才能进行。各个相关部件的迟滞的结果就是集成过程的迟滞和所有后续测试步骤执行的迟滞。

所以，只有在所有部件完成整车系统组合时，例如 ECU、设定点发生器、传感器、执行器和车体都已经准备就绪，ECU 软件功能测试才能进行。实验室汽车的应用使得 ECU 早期测试并不需要真实环境部件就可以实现。这种方式同时也可以降低测试驾驶员或汽车

原型暴露在一定风险中的可能，预防意外事故的发生。

这种虚拟的有效性检验将会越来越具有重要的意义。然而，给定功能的最终有效性检验，即测试是否满足了用户需求，进行这个测试的唯一出发点是用户自身的观点，也就是需要进行实车的验收测试。

1.5.3.2 软件功能复用

降低风险的第二个方式是软件功能复用。成功复用的先决条件是一个清晰的整个系统的模块化。在今天如果把复用的目标指向软件的源代码层复用，那么将来将会阻碍新一代的系统和软件体系的引入，也会阻碍软件在新一代硬件平台上的移植。

然而，只有在模型级别上，才能真正认识到复用的显著优势。通过复用验证的功能和环境模型，以及通过复用测试的案例、复用模拟仿真校准数据、复用实验室/测试台以及试验汽车，都能最大地降低风险。

1.5.4 标准化和自动化

在功能开发中，应用标准化和自动化有助于节省成本和改进质量。

1.5.4.1 标准化

过程的标准化和测量、校准、闪存设计和诊断工具描述格式的标准化首先在 ASAM[18] 和 ISO[24,25]中出现。这些标准广泛应用在汽车行业。图 1-21 大概描述了已经核准的工具和接口标准 ASAM-MCD1b、2、3 的软件体系结构。

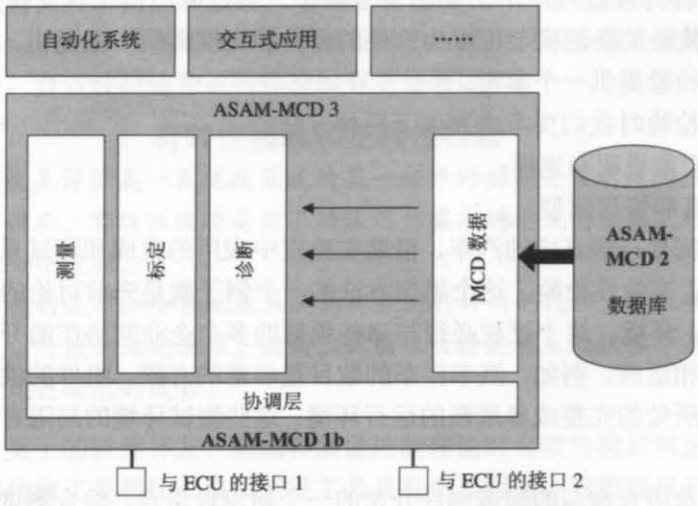


图 1-21 面向 ASAM 标准和工具的软件体系结构

同时，我们也成功引入了用于 ECU 的面向微控制器的软件结构体系标准化方法。举个例子，实际的软件功能，比如我们所说的应用软件的控制和监控功能（应用层软件），与部分依赖于硬件平台系统层软件功能之间存在着区别。

系统层软件也包括了车载内部间通信及与车外部通信所需要的软件。由于系统层软件

和应用软件的区别，应用软件功能可以很大程度上不依赖于硬件来实现，这也促进了应用层软件功能在不同微控制器间的移植。

这些包含与微控制器 I/O 单元硬件相关方面的软件成分聚合在一起，也就是所谓的系统层软件的硬件抽象层（Hardware Abstraction Layer，HAL）。如图 1-22 所示，通过总线与其他系统通信所需要的输入/输出单元并不包含在 HAL 中，对所需的总线驱动模块是分开独立考虑的。系统层软件也包括那些通过网络与其他控制器通信的软件组件，或是与工具如诊断测试装置的通信所需的软件组件。

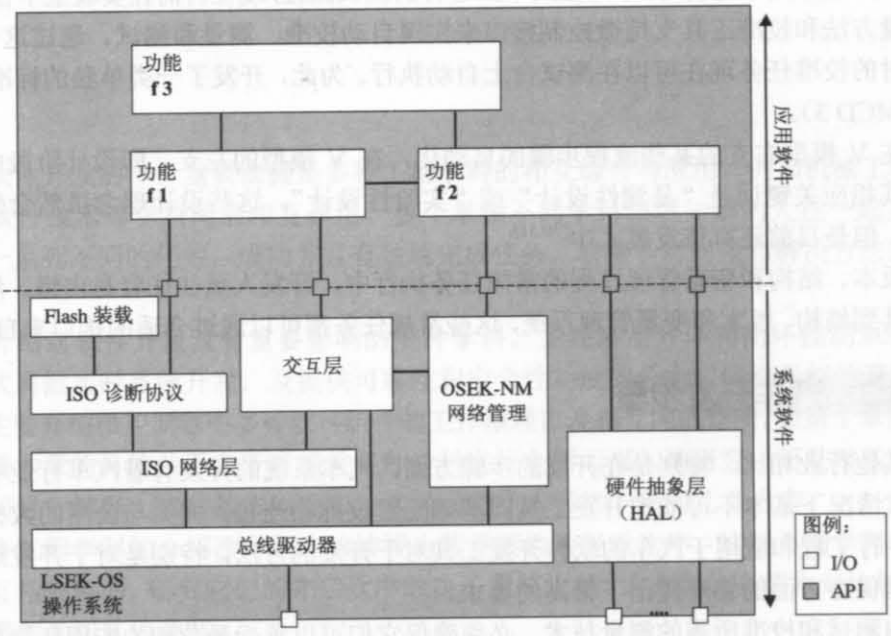


图 1-22　面向微控制器和 OSEK/ISO 标准的软件体系结构

标准化的软件组件的例子有很多，比如基于 OSEK[16]标准的实时操作系统、通信和网络管理，以及基于 ISO[25, 26]标准体系的诊断协议。

目前，标准化软件大体集中在那些从属于系统层软件的组件上，而汽车制造商和汽车供应商也发现它们并没有很大的竞争意义（图 1-22）[27, 28]。这些软件组成提供了标准的应用程序设计接口（Application Programming Interface，API）。因而，应用软件功能可以在很大程度上不依赖于硬件而独立开发。本书第 2 章会就软件结构体系和微控制器标准组成进行详细讨论。

所有系统层软件组成的完全标准化对于节省成本和质量控制有着深远的潜在作用。然而，由于应用软件功能通常能带来很大的竞争力，汽车制造商和供应商对于开放标准的引入并没有多大的关注。尽管如此，一旦确定下责任问题和版权保护，整个企业范围内的标准化，比如，功能库的建立，可以给双方都带来巨大的获益。到那时，汽车制造商可以根据不同的供应渠道来展开软件功能的开发。供应商则可以开展面向所有客户的标准软件功能开发。

1.5.4.2 自动化

容易出错的程序任务的自动化为软件的功能的开发提供了巨大潜力。自动化所达到的高可重复性带来可观的成本和时间效益，更重要的是改善了产品质量。

越来越多的汽车制造商和供应商在以下开发步骤中使用自动化：

（1）通过快速原型工具进行功能原型的生产；

（2）ECU C 代码生成器（图 1-20）。

随着实验室汽车的使用，原先在汽车上进行的测试流程，现在转而在实验室中自动实现。

测量方法和校准工具支持微控制接口来实现自动校准、测量和测试。通过这个接口，原本耗时的校准任务现在可以在测试台上自动执行。为此，开发了一类单独的标准（比如，ASAM-MCD 3）。

处在 V 模型右支的某些流程步骤的自动化，在 V 模型的左支，即设计阶段中就被考虑到。其相应关键词是“易测性设计”或“实验性设计”。这些设计理念虽然会带来巨大的收益，但是目前还有待发展之中[30,31]。

在版本、结构和变量管理过程的常规任务执行中，开发人员往往容易出错。然而，从开发工具到结构、版本和变量管理系统，这些常规任务都可以通过合适的接口来自动执行。

1.5.5 汽车开发步骤

与其他行业相比，特别是在开发的步骤方面，汽车系统的开发有着汽车行业中的特殊性。通常情况下基本不可能把开发工具同基础配置设施相连接，例如与公用的数据网络相连接。并行工程和应用于汽车的软件开发工具对于开发的方法，特别是对于开发结果的管理和数据保存方面的管理提出了很高的要求。

车内测试和校准所需的测量技术，必须确保它们可以承受最苛刻的应用环境条件，其中包括极高或极低的温度、湿度，电磁兼容性，变动的电压供给，振动以及狭小的安装空间。另外，车内测量技术的用户界面也必须适合在车内使用。

在汽车开发中，为了保证对由 ECU、设定点发生器、传感器和执行器构成的系统来实现的汽车功能的有效性，汽车测量技术必须比用于车载 ECU 传感器系统的测量技术具有更高的性能级别。特别是为了能够及时地获得 ECU 内部的信号，连接 ECU 和测量工具的数据接口必须能够以较高的速率来传输数据。

第 2 章

Chapter 2

基础知识

汽车软件开发的一个重要前提是多种工程学科的相互综合与应用。例如机械工程、电子工程和软件技术等学科间的相互作用。多个掌握不同学科领域知识的人在一起共同工作，完成一系列不同的任务。成功并且有效地完成任务，取决于对问题和解决方法的正确认识。

本章介绍对软件开发具有重要影响的多种学科。主要涉及开环和闭环控制系统的开发，嵌入式离散实时系统开发，及提供可靠性和安全性功能的分布式网络系统的开发。

本书主要介绍微控制器中多种软件组件的工作原理以及相互间的作用，如第 1 章图 1-22 所示。编撰本章的目的并不在于对相关多种学科进行全面而深入地研究，而是对基本原理和技术名词进行解释，这些基本原理和技术名词贯穿本书接下来的几章内容。

对各个工程学科的介绍次序不应该被视为其重要性的轻重次序。这些不同学科在一定程度上是互相依赖的，研究它们的先后次序取决于系统本身。这方面的介绍本章比较全面，读者并不需要参阅其他章节。

2.1 开环和闭环控制系统

车辆传动系统、底盘和车身子系统的多数功能是执行一些控制任务。因此，深入了解与控制技术相关的方法、术语和概念是设计这些功能所必需的基础。

2.1.1 建模

在控制系统和监视系统功能设计的初始阶段，建模被视为技术实现的抽象化。抽象化的任务（也就是建模）是建立一个数学模型。为控制和监视系统建模（即开环控制模型和闭环控制模型）与为被控制和监控对象的建模是有区别的（即开环对象模型和闭环对象模型）。

实现控制任务所选择的控制方法，与被控制对象的不同物理特性是无关的。影响开环

和闭环控制系统设计的首要的决定性因素是被控对象的静态特性和动态特性两个方面。本书设法通俗易懂，全篇采用汽车行业相关控制单元专门术语。这些名词描绘了除实现其他功能之外，还实现开环和闭环控制任务的系统和装置。

至于被控的具体物理参数的类型，如温度、电压、压力、扭矩、输出动力和转动速度，及系统和装置的技术实现是次要的。

抽象化可能使开环和闭环控制系统控制技术发展成为一门单独的工程学科。而事实上，这一工程科学分支试图刻画不同具体物理系统的共同的系统特征。随着控制系统普遍应用设计方法的相关理论基础的提出和发现，控制学科作为多种工程学科的交叉学科已经发展成熟。

2.1.2 结构图

在很多情况下，建模可由可视化的图表来表示，一般采用结构图。这些结构图描绘了单个组成部分的响应特性，同时也描绘了一个系统内各组成部分之间产生的信号。图 2-1 表示开环和闭环汽车控制功能逻辑系统体系结构图，清楚地解释和定义了闭环和开环控制技术的一些基本术语。

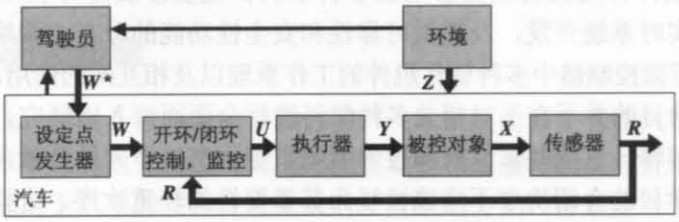


图 2-1 汽车开环和闭环控制系统功能的结构框图

闭环控制任务就是通过检测控制变量 X（即调节变量），然后被控变量与参考变量 W 相比较。根据这一比较结果，调整变量 X，使其接近参考变量 W。这一过程通过一个闭环回路来完成，即所谓的控制环。闭环控制的目的是使控制变量 X 的值接近参考变量 W，尽管存在由于干扰变量 Z [32]所导致的干扰情况。

相应地，开环控制任务是一个系统的一个或多个输入变量影响某个输出变量使其符合系统的设计特征的过程。其控制操作的特点是包括单一转换元件或包括整个开环控制环的一系列开环动作序列[32]。

开环和闭环控制模型以结构图的形式表示系统的构成，用方框表示转换环节，箭头连接方框表示信号流方向。一般情况下，汽车上应用的控制系统是属于多输入/多输出系统的范畴。因此，处理信号都是矢量形式的（图 2-1）。下面列出这些不同信号：

被测变量或反馈变量	R
开环/闭环控制输出变量	U
参考变量或设定值	W
驾驶员设定值	W^*
开环/闭环被控制变量	X

被操纵变量	Y
干扰变量	Z

根据组成功能模型的方框，可分为开环/闭环控制器模型、执行器模型、被控对象模型、设定值产生器和传感器模型、驾驶员、运行环境。驾驶员通过设置设定值来影响其中一个或其他控制系统。获取驾驶员设定值（如驾驶员对于开关和踏板的操作）的器件，也称为设定点发生器。相应地，传感器捕获控制部分或被控对象产生的信号。

由此得出结论，通过标准操作，即通过设定点发生器或是转换上层系统产生的默认参数值，系统能够引入参考变量或设定变量 W。一般情况下，系统是递阶分层次结构的。

由于这种控制和调节模型的建模方式是对于具体的物理实现的抽象，因而这种模型也适合对于电子系统的软件部分所实现的控制和调节功能的建模，也适用于描述他们与设定值发生器、传感器和执行器等各电子控制器的软件之间相互作用，以及与其他汽车组成部分和车载电子系统的相互作用。鉴于以上这些优点，开环/闭环控制建模的结构图建模也广泛应用于软件实现的汽车功能的开发中。因此，尽管这种方法忽视软件的根本方面，但还是作为软件协调过程的连接要素。

举例：PI 控制器结构图

结构图同时也用以表示闭环控制回路中的某个环节，如例子中的控制器。图 2-2 表示一个由两个环节组成的控制器的结构图。其中一个环节具有比例响应特性，另一个环节具有积分信号操纵特性。因此，这种控制器被称为 PI 控制器（比例加积分控制）。

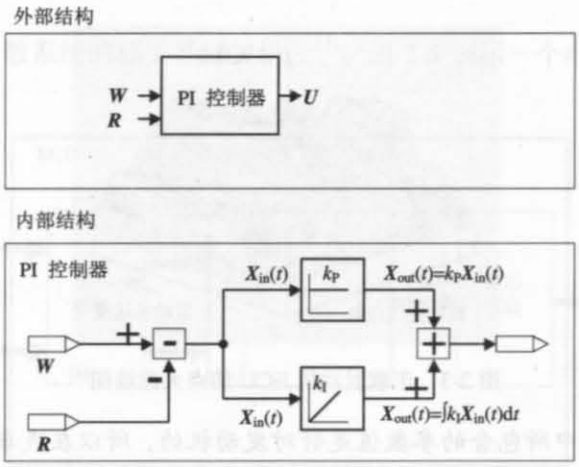


图 2-2　PI 控制器的结构框图

任何一个控制器的典型特征是控制器输入变量 W 和输入变量 R 的比较。图 2-2 所示的 PI 控制器，是通过比较两个输入变量，也就是比较参考变量 W 和反馈变量 R 之间的差异来实现的。所获得的比较结果，即系统偏差，作为控制器两个环节的输入变量。

比例环节，其响应特性

$$X_{out}(t) = k_P X_{in}(t) \tag{2.1}$$

积分环节，其响应特性

$$\boldsymbol{X}_{\text{out}}(t)=\int k_{\text{I}}\boldsymbol{X}_{\text{in}}(t)\,\mathrm{d}t \tag{2.2}$$

放大因素由控制器参数 k_{P} 和 k_{I} 给定。PI 控制器两环节的输出相加作为控制器输出变量 $\boldsymbol{U}$。

因此，PI 控制器的转移函数表示式为

$$\boldsymbol{U}(t)=k_{\text{P}}(\boldsymbol{W}(t)-\boldsymbol{R}(t))+\int k_{\text{I}}(\boldsymbol{W}(t)-\boldsymbol{R}(t))\mathrm{d}t \tag{2.3}$$

结构图分别给出了外部视图和内部视图（图 2-2）。

对于控制系统回路中所有环节的这一类型建模，构成了系统分析、设计、实现和测试方法的基础。

在这里并不打算对于系统的建模、分析和设计的方法和过程进行更为深入、详细的讨论，相关内容请参阅文献[33]~[36]。

决定控制功能特性的决定性因素一方面是传递函数，即控制算法，另一方面是相关控制参数的设置。给定控制器功能的各种参数组成的一组数，称之为参数集。在一些汽车功能控制参数的利用上，除了利用上面提到的实例中的标量 k_{P} 和 k_{I} 之外，也可以利用特征曲线和三维特征图来代替控制参数。

举例：点火曲线图

这方面的一个例子是车载发动机 ECU 的点火曲线图（图 2-3）。根据发动机的当前工作点（也就是根据发动机速度的输入参数和负荷以及相对充气负荷），点火曲线图给出了相对于发动机油耗和废气排放特性的最佳点火角度[4]。

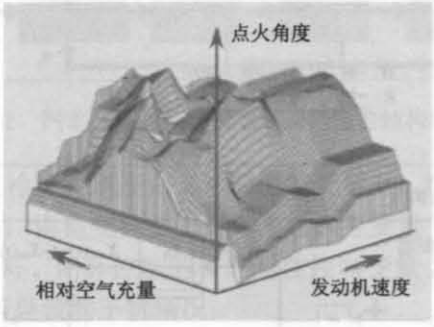


图 2-3 车载发动机 ECU 的点火曲线图[4]

因为点火曲线图中所包含的参数值是针对发动机的，所以在汽车的开发改进阶段必须明确和调整这些参数值。

正如在控制设备的讨论中所提到的，不管控制功能是通过机械系统、液压系统、电气系统来实现，还是通过电子系统来实现，从控制技术的角度来看都是次要的。例如，根据所应用的技术，完成实现如图 2-2 所示的 PI 控制器可以有很多种方法，这些方法完全不同。然而，我们不得不承认汽车制造领域是一个需要不断努力的领域。控制功能的实现通过 ECU 并结合机械、电子、液压环节的协作来实现，在可靠性获得、汽车总重、必要安装空间及成本等方面拥有许多优点。因此，这种实现形式成为首选形式。从而，接下来的部分我们将更详细地讨论 ECU 的结构及其应用原则（图 2-4）。

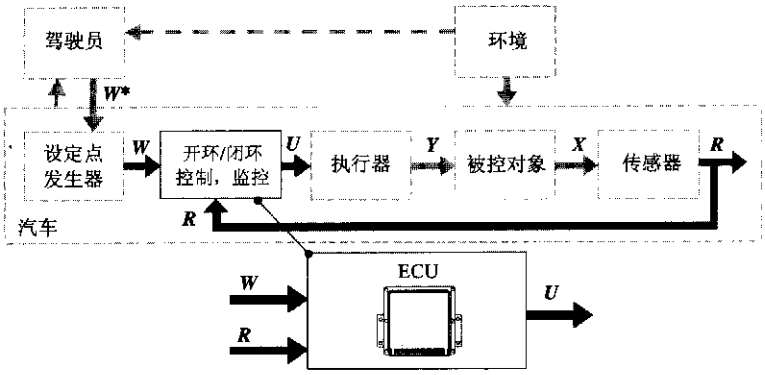


图 2-4　在 ECU 中实现开环/闭环控制功能

从在电子控制仪器中应用的微控制器应用软件开发观点来看，开环/闭环控制器是功能模型，设定点发生器、传感器、执行器、被控对象、驾驶员和运行环境是外围模型。

2.2　离散系统

相对应于机械、电子和液压环节中的模拟信号处理，在电子系统中应用的数字微处理器对 ECU 的输入变量加以离散化处理。这就意味着，开环/闭环控制功能是由离散化来实现的。

这一部分介绍离散系统的相关基础知识[37,38]。图 2-5 表示一个电子控制器的简化结构图。

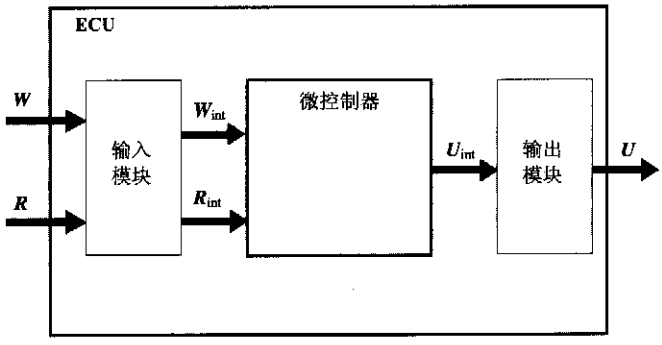


图 2-5　含开环/闭环控制系统结构框图的 ECU 模型

一旦设定点发生器和传感器获得外部输入信号 W 和 R，输入信号首先经过 ECU 的输入模块进行处理，生成离散信号，作为微处理器内部输入变量 W_{int} 和 R_{int}。同样地，输出模块将微处理器的内部输出变量 U_{int} 转变为执行器所需的外部输出连续信号 U。一般地，输入和输出模块包含信号调节和增益电路。内部信号对于一个微控制器中的软件开发是有意义的。因此，为方便起见，本文接下来不再区分内部信号和外部信号。除非另有规定，下文所提到的内部信号将表示为 W、R 和 U。

2.2.1 离散时间信号和系统

在模拟系统中，所有出现的信号都是连续时间函数。因此，当相对于每个时刻 t，都有确定的状态值 $X(t)$与之对应（图 2-6（a））。这种信号称为连续时间且连续值信号。

而如图 2-6（b）所示类型信号 $X(t)$，只在确定的间断时刻 t_1、t_2、t_3等处测量或是“采样”，所以称为离散时间连续值信号或采样信号。其表达式为

$$X(t_k) = \{X(t_1), X(t_2), X(t_3), \cdots\} \tag{2.4}$$

其中，$k = 1$，2，3，…

时间间隔 $dT_k = t_k - t_{k-1}$ 称为采样频率。在所有的采样点，采样频率可以相同，也可以不同。

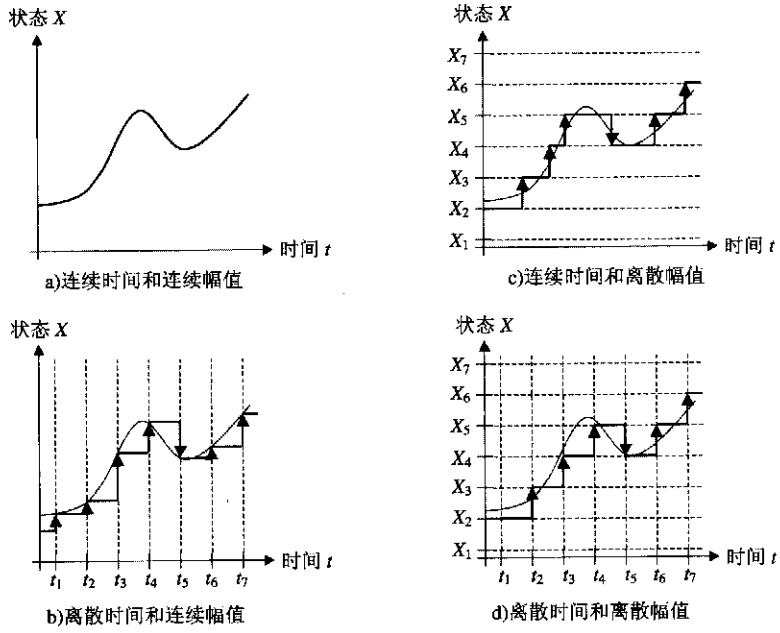


图 2-6 连续信号的采样模型

举例：发动机 ECU 采样频率

发动机 ECU 执行一些独立功能。使用传感器获得发动机状态和驾驶员指令，控制发动机专用执行器。

点火和喷油两个基本功能必须使发动机执行器运行与相应曲轴位置同步。发动机速度的变化也会改变这些功能的采样频率。

然而，其他功能，如通过踏板位置依靠踏板动作传感器来获得驾驶员指令，能够以恒定时间间隔执行（也就是采样频率不变）。

采样频率 dT 是离散时间系统的一个基本设计参数。所要求的采样频率由被控对象的动态特性来决定。一般来说，以离散时间控制器来控制连续时间被控对象时，采样频率取为控制系统中主要时间常数最小值的 1/10 到 1/6 之间[34]。采样频率 dT 对于离散时间控制

系统的行为产生很大的影响。如前面所举的例子中，一个控制单元在同一时刻执行多个控制功能是非常常见的，且每个功能的采样频率是不同的。

如果所给系统至少出现一个离散时间信号，那么就称为离散时间系统。例如，对于微控制器，其输入信号就是离散时间信号。

2.2.2　离散值信号和系统

模数转换器（简称 A/D 转换器）通常用于输入信号的获取，称为采样元件[34]。设备的有限字长要求幅值量化（即形成离散值信号）（图 2-6（c））。

幅值量化产生非线性效应。例如，在模数转换过程中，由于 $\boldsymbol{X}_{\min}$ 和 $\boldsymbol{X}_{\max}$ 值域限制，而产生非线性特性，值域范围内 $\boldsymbol{X}(t)$ 设为离散值 $\boldsymbol{X}_i$ ：

$$\{\boldsymbol{X}_1,\ \boldsymbol{X}_2,\ \boldsymbol{X}_3,\ \cdots,\ \boldsymbol{X}_n\} \tag{2.5}$$

其中，$\boldsymbol{X}_{\min} \leqslant \boldsymbol{X}_i \leqslant \boldsymbol{X}_{\max}$。

$\boldsymbol{X}(t) - \boldsymbol{X}_i(t)$为量化误差。

同样地，控制单元输出信号处理过程类似，称为数模转换。这里，输出调制信号脉冲宽度有多种形式。针对本文，所有输出离散信号都采用数模转换（简称为 D/A 转换）。在 D/A 转换过程中，给定值 $\boldsymbol{X}_i$ 保持不变直到下一个采样周期。相应地，D/A 转换器称为保持元件[34]。

2.2.3　离散时间且离散值信号与系统

如果时间和幅值都离散化，则称为离散时间且离散值信号（图 2-6（d））。

如果给定系统至少出现一个离散时间且离散值信号，则称为离散时间且离散值系统或数字系统。

ECU 微控制器所执行的程序的输入变量都是离散时间且离散值信号。如图 2-7 所示，在闭环控制回路或开环控制系统中，微处理器可用一个方框来表示。

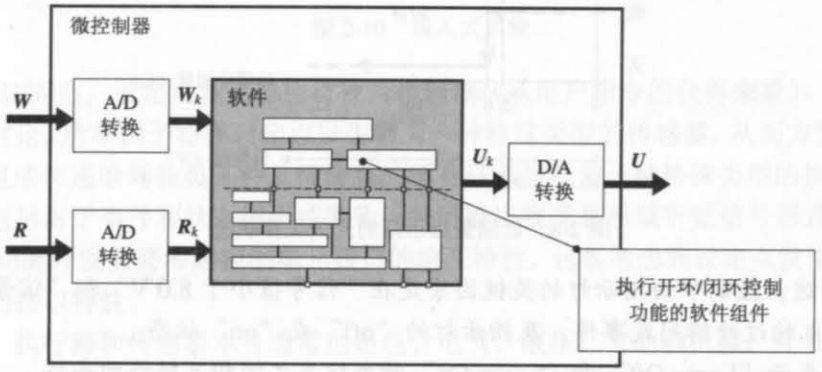


图 2-7　含开环/闭环控制系统结构框图的微控制器模型

图 2-7 中，普通连续时间且连续值输入信号 $\boldsymbol{W}$ 和 $\boldsymbol{R}$ 转换后生成离散时间且离散值信号 $\boldsymbol{W}_k$ 和 $\boldsymbol{R}_k$。接着，经程序处理生成离散时间且离散值输出信号 $\boldsymbol{U}_k$，再经过转换后生成

连续时间且连续值信号 U。开环和闭环控制功能响应特性及各控制参数，必须由微控制器中的软件部分实现。

2.2.4 状态机

物理变量一般包含连续时间且连续值信号，连续时间且连续值系统响应特性可以由微分方程来表示，而离散系统响应特性由差分方程表示。

时间和幅值离散化，把某一事件简化转换为一个个离散状态点 $X(t_k)$、$X(t_{k+1})$。可能的或相关的状态点个数，及可能的或相关的事件个数在多数离散系统中都是有限的。建模中状态机的开发使用要考虑这一现实因素。

举例：低油量指示灯控制

油位传感器测量车辆油箱的油面高度，生成 0~10V 范围内的模拟信号（电压值与测得油面成比例）。该模拟信号作为控制节油指示灯的输入信号。并以离散时间且离散值形式在仪表板上显示。

在这一处理过程中，8.5 V 模拟信号表示油箱里还剩余 5L 燃油，10 V 信号值表示燃油用完，0 V 信号值则表示油箱装满。因此在信号值高于 8.5 V 时，节油指示灯必须导通并给出警告。

为避免由于油箱中油量时刻变换而引起的指示灯快速连续开关产生的闪烁现象，必须要有一个滞后功能。在油量多于 6L，相应的信号值低于 8.0 V 时，关闭节油指示灯是比较合适的。相关的开关操作如图 2-8 所示。

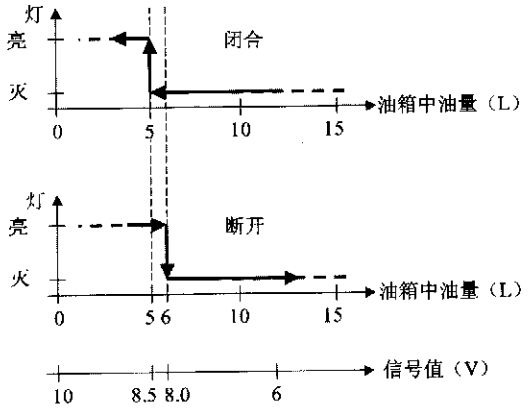


图 2-8 低油量指示灯的开关工作状况

综上所述，控制节油指示灯的关键因素是在“信号值小于 8.0 V”和“信号值大于 8.5 V”两临界点的过冲瞬间或事件，及指示灯的“off”或“on”状态。

图 2-9 表示“Lamp Off”和“Lamp On”两个状态之间相互转换的条件。

这一类型的状态转换图，也称为状态机，作为描述离散系统的图表符号，也经常用于模拟离散的汽车功能。

更多关于连续和离散信号与系统，请参阅参考文献[37，38]。

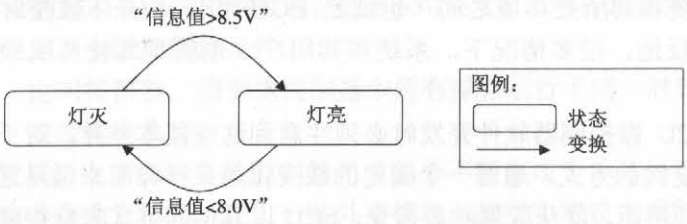


图 2-9　低油量指示灯的状态/变换图

2.3　嵌入式系统

电控单元、设定点发生器、传感器和执行器构成一个能够影响被控对象状态的电子系统。多数情况下，ECU 作为驾驶员-车辆-环境整个系统的一个组成部分，是看不到的。例如，如果 ECU 专门用于控制器功能，就不会配置任何用户接口。一般情况下，这是传动系统、底盘和车身 ECU 应用的普遍特征。驾驶员和车辆所有者对 ECU 一定程度上的影响，在很多情况下是间接的，在某些场合下是不存在的。通过用户接口获得参考变量基本上是间接实现的，一般也是受到限制的（图 2-10）。具有以上所描述特征的系统称为嵌入式系统。

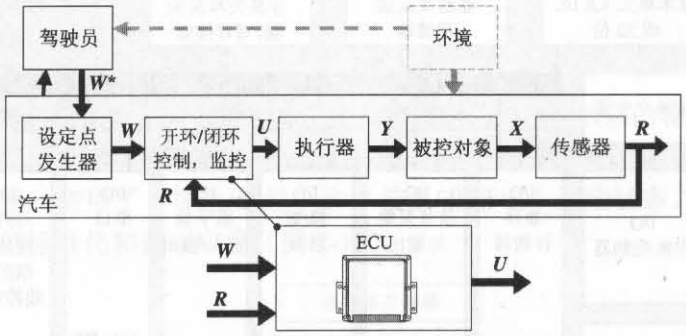


图 2-10　嵌入式系统

就 ECU 而言，设定点发生器也可视为传感器（从用户指令的获得来看）。因此，接下来部分的讨论，所举例子将视设定点发生器为一种特殊类型的传感器，从而方便读者理解。类似地，组成传递给驾驶员某种反馈信息的所有环节被视为一种特殊类型的执行器。这些反馈信息包括对于事件和状态提示或警告，并可能以视觉显示或听觉信号形式出现。

ECU 功能开发需要考虑控制单元接口的响应特性，也需考虑到设定点发生器、传感器和执行器的接口特性。

其次，执行器和传感器本身通常也是包含电气、液压、气动和机械，并且越来越多是由电子组件来构成的系统。如果在执行器和传感器中还包含信号的前处理或者后处理部分中，通常称为智能执行器或智能传感器。

响应特性一方面包含随时间变化的动态响应，另一方面包含静态响应，如传输信号的值域及物理分辨率。

在嵌入式系统和其所处环境之间（也就是 ECU 和闭环/开环被控对象之间）总是存在直接接口。相反地，很多情况下，系统和其用户（也就是驾驶员或乘客）之间只有间接接口。

因此，在 ECU 微控制器软件开发时必须注意到这些基本差异。对于被控对象的动态特性，以一种确定性的方式，通过一个固定的或变化的采样频率来循环采样，获得被控对象的当前的状态。然而，就获取驾驶员指令，ECU 以其他的方式来获得驾驶员的输入更加具有优势。例如，在操作电动车窗的开关时，驾驶员指令应被视为某一事件，该事件偶尔发生一次，但要求即时响应。

一般来说，微控制器要求能够处理周期和非周期事件。因此，对于所有参与功能开发的开发者来说，理解掌握 ECU 中应用的微控制器结构、工作原理、接口和设计是重要的前提。

2.3.1 微控制器结构

一个微控制器由以下环节构成，它们之间相互作用形成整体（图 2-11）[39,40,41]：

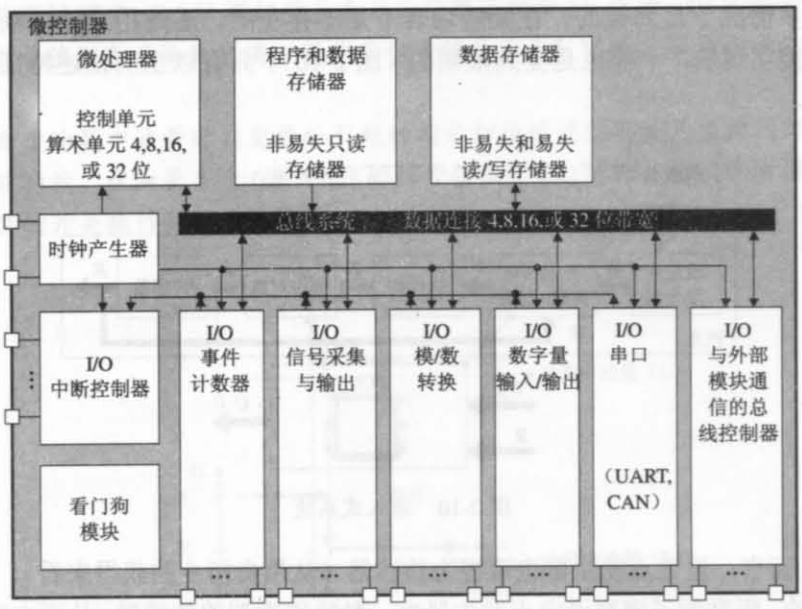


图 2-11 微控制器结构图

中央处理器（CPU） 微处理器自带控制单元及算术逻辑单元（ALU）。后者处理算术和逻辑操作，而前者确保从程序存储器中所获得指令的执行。这一分工适合各种实际应用。

输入输出模块（I/O 模块） 处理外围数据传送。包括输入/输出设备，程序中断控制电路和总线系统，及其他通信控制设备，如 CAN[2]。

程序和数据存储器 非易失永久存储器。存储程序，如开环和闭环控制算法，及恒定参数集，如开环和闭环控制参数。其理想技术是非易失存储技术。该存储器中程序和参数集存储于不同存储区域。因此，我们通常使用程序和数据存储器。

数据存储器 存储程序执行生成的数据。因为这一特征，该存储区域也称为随机存取存

储器（RAM）。其理想技术是读/写存储技术。根据应用要求，使用易失或非易失读/写存储器。

总线系统　互连单个微控制器元件。

时钟发生器　也叫振荡器。确保微控制器中所有操作运行于同一时钟频率。

看门狗模块　监控程序执行。

随着技术的发展，上面所述各微控制器组成部分不断集成于单片处理器芯片中。这使得微控制器能作为一个独立单元。连上一些外接模块，就能实现设定功能。例如外接扩展存储器。因此，有内部存储器和外部存储器之分。

2.3.2 存储技术

讨论了程序存储器与数据存储器间的不同要求后，这一部分将关注各种半导体存储器技术。

半导体存储器用于存储以下内容：

（1）数据，如 I/O 数据、状态及中间结果，这些都要求有快速读写存取能力；

（2）执行程序，一般要求永久存储；

（3）恒定参数集，一般也要求永久存储。

存储器存储包括以下操作：

（1）写入；

（2）暂时或永久存储；

（3）恢复和读取信息。

半导体存储器利用了电子元器件容易产生和辨别的两种不同状态的物理效果。半导体存储器的优点在于其与微控制器其他元件间的技术兼容性，及这种联合提供的多种综合选择。

为了存储信息，应用了“导电/绝缘”和“带电/非带电”这些状态。接下来的部分讨论主要技术，包括标准化和基本应用（图 2-12）。

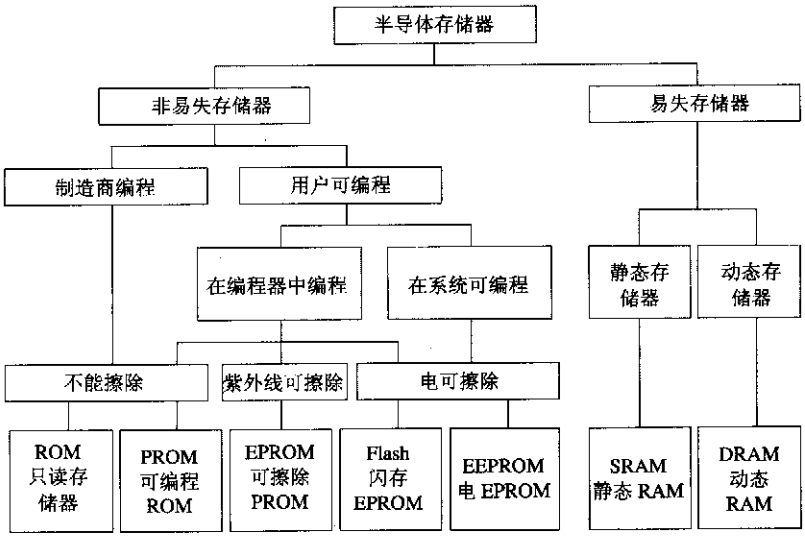


图 2-12　存储器技术概况

根据各自应用，半导体存储器以位（bit，比特）或字（word）为存储单位，这里的“字”表示位的一个逻辑集合，以适应微控制器的并行处理。因此，字长等于并行处理的位数。微控制器通常提供4、8、16、32或64位字长。8位称为1字节（byte）（1 byte = 8 bit）。

2.3.2.1 读/写存储器

RAM

随机存取存储器（RAM）为任一主存储单元提供直接存取。一有需要，信息就能够从RAM中读取和写入RAM。主存储器使用易失RAM，这意味着出现负载功率时，存储器内容就会丢失。在随机存取存储器中，包括静态RAM（SRAM）和动态RAM（DRAM）[39]。

静态RAM只能写入一次，且写入内容只能保持到下一个工作电压出现。因为泄漏电流会造成动态RAM存储内容丢失，因此必须定期更新。

同样，在RAM的电源部分集成一个附加备用电源以方便非易失数据的存储。这种非易失RAM用符号NV-RAM来表示。

2.3.2.2 非擦除只读存储器

只读存储器（ROM）为任一主存储单元提供直接存取。然而，顾名思义，其存储内容只能读取而不能写入。

ROM/PROM

ROM是非易失存储器。其存储内容在无工作电压时仍能保留。ROM通常包含程序代码，如开环和闭环控制功能算法；以及常数，如相关参数集。这些信息能够随时存取，可以由制造商编程写入存储器——产品生产的最后一步，也可以由用户通过专门程序编程写入特定存储模块。这些可编程只读存储模块也称为可编程ROM（PROM）。

2.3.2.3 可重复编程非易失存储器

一些类型的只读存储器，其存储内容可以擦除和重复编程。包含以下几种类型：

1. 可擦除PROM（EPROM）

可重复写入只读存储器在紫外线下能完全擦除，从而可以重复写入。然而值得注意的是，这一处理过程需要专门的设备，且花费大。

2. 电EPROM（EEPROM）

电EPROM（EEPROM）也称为E^2PROM。这一可重复写入只读存储器可以电擦除和重复编程。擦除和重新写入操作既可以由其他配置器也可以由ECU自身来完成。EEPROM使存储器的每个独立信道都能重新写入。

因此，这一存储技术也应用于非易失数据存储器。在发动机管理中，一旦发动机关闭，相应的控制参数就会被存储起来。另一个应用例子就是将检测故障数据存储在故障存储器中。故障存储器结构详细讨论见第2.6节。EEPROM也可以用于存储生产中各种控制所要求的软件参数，及汽车服务程序。更多实用程序详见本书第6章。

3. Flash EPROM

Flash EPROM（有时简称为Flash）包括下一代EPROM和EEPROM。这一存储器通过电脉冲来擦除整个存储区域或闪存全部存储内容。一旦被擦除，该区域就可以重新编程。

Flash 存储器编程可以使用一个编程器来完成。然而，闪存技术的绝对优势在于 Flash 存储器允许使用编程工具进行系统内编程，甚至在车载 ECU 内。因此，闪存技术应用于一些需要大量数据非易失存储且在产品寿命周期里进行改写的场合中（例如在 ECU 中作为程序或数据存储器）。Flash 编程步骤在本书第 6 章进一步讨论。

2.3.3　微控制器编程

一个微控制器中微处理器的执行程序通常永久地存储在非易失存储器中。在处理不同应用中，存储程序并不会被修改。只有当新的程序版本被烧入时，Flash 存储器中软件程序才得到更新。

这一部分考查微控制器编程细节。这里所谓的软件包括微控制器驱动系统中存储器存储的全部程序和数据。这些程序由微处理器执行。

这意味着软件工程中，必须执行以下规范，如程序代码要符合微处理器的执行，及参数和数据集存储于微处理器的数据存储器中。

2.3.3.1　程序型式和数据型式

在以下的讨论中，程序代码将称为程序型式，其必须被下载到微处理器程序存储器中。参数集将被称为数据型式，其必须被下载到微处理器数据存储器中。

为了简化，经常提到控制单元软件或是 ECU 程序。然而，注意到给定 ECU 的结构有可能由多个微控制器构成（也就是功能计算器和监控计算器）。因此，微控制器软件能提供更高的精确度，这就是微控制器软件程序型式和数据型式的优点。

2.3.3.2　微控制器功能原理

关于编程，首先来看图 2-13 所示的微控制器简化模型。如图 2-13 所示，微控制器包括微处理器、指令存储器（也称为程序存储器）和输入/输出模块[39]。它们之间通过总线交换数据和控制信号。

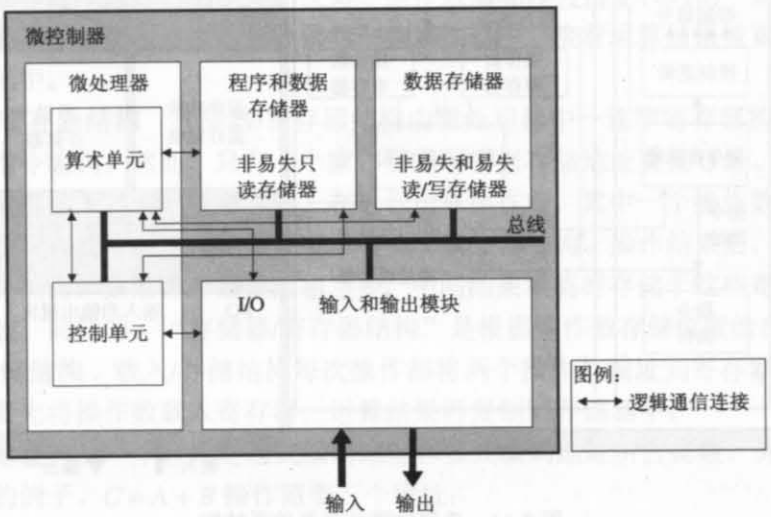


图 2-13　微控制器的简化结构图

微处理器包括可编程机构，处理地址和数据信号及程序的特定时间点控制和逻辑执行。各存储区域存储数据和程序指令。读/写存取存储器（例如 RAM）存储可变数据。适合存储程序指令和常数的是只读存储器（例如 ROM）。大多数微处理器还包含一个容量较小的综合存储器模块，称为寄存器。目的是提供快速地读写存取功能。

输入输出模块（I/O 模块）用于接收外部信息和将数据传送至外围设备。为了便于应用它们的功能，输入/输出模块提供了有限的编程操作。I/O 模块的典型例子是用于数据输入的模数转换器，及处理数据输出的脉冲宽度调制模块和数模转换器。计数器/定时器用于计算外部脉冲或测量事件时间间隔。外部器件和（或）其他微控制器间通信可通过连续且并行接口执行。例如，多个微控制器间数字数据通信运用 CAN 总线[2]。根据应用要求，微控制器能够扩展附加功能。

2.3.3.3 主要微控制器操作

图 2-13 所示方框图给出了微控制器的主要操作：

（1）数据处理；

（2）数据保存；

（3）外围数据交换。

以上这些主要功能，使得微控制器能应用于数据的传送、保存和处理。接下来的部分我们来更详细地了解单个微控制器实现这些操作的结构单元。

2.3.3.4 微处理器结构和指令系统

微处理器处理通过输入模块输入的数据并控制数据流程。微处理器的寄存器存储操作数、操作结果和地址，如图 2-14 所示为微处理器结构[39]。

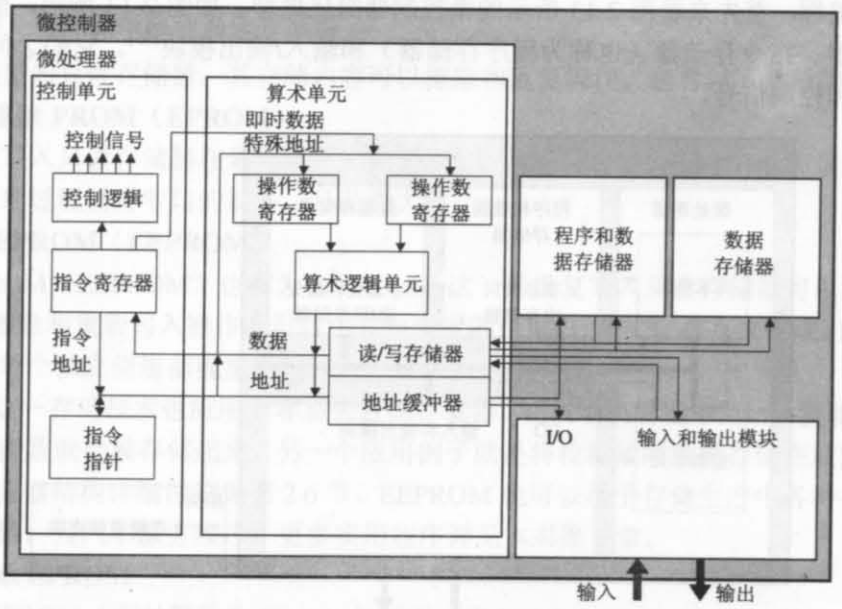


图 2-14 典型的微处理器体系结构

在这一简图中，为了表示清楚，我们省去了增加处理速度的扩展设备。

该结构可以通过编程器所有寄存器数量来表示。

设定特殊控制寄存器，就可以进行明确配置。因此，控制寄存器也可以视为准静态指令系统扩展。例如，中断控制寄存器定义中断事件，即允许中断和禁止中断。另外控制寄存器还可以用于定义算术逻辑单元（ALU）和 I/O 模块功能。

一些操作可能会影响微处理器程序处理过程。例如，如果收到外围的一个中断请求，就会根据定义的存储地址执行子程序。当处理到中断服务程序时，只有较高优先级的中断请求才会被执行。

所有其他中断请求被暂时存储于存储器中，只有当前中断服务程序执行完毕后才进行处理。处理过程中产生的状态信息可以存于中间存储器。然而，这可能会在一定情况导致指令过长。因此，状态寄存器和特殊寄存器（除控制寄存器外）都被集成到微处理器中。这组状态寄存器也包括程序状态寄存器、中断状态寄存器和乘法器状态字。如果这一类型中断逻辑以硬件解决方法产生，则称为硬件中断系统。

通常，为了减少微处理器存储器的读/写操作，一些特殊计算寄存器，即累加器，就会集成入微处理器中。这样，中间结果和经常使用的变量就可以在微处理器内部进行处理。这种减少读/写操作的方法，使得时钟频率增加，同时降低微处理器电流蠕变影响。

1．操作数存储器

操作数是算术逻辑操作的对象。为了确保算术操作前后操作数快速装载，有多种方式可供选择。操作数存储单元是微处理器结构的决定性因素，以下为一些典型结构。

存储器/存储器结构　存储器/存储器结构通过主读/写存储器（RAM）提供快速操作数装载。通过连接操作数存储地址和算术操作结果来实现。例如使用这种方法，两个存储于 RAM 中的操作数，通过一条简单指令就可以进行相加。相加结果可以直接写回 RAM 中。这种“存储器/存储器结构”是根据操作数存储位置给出的。

累加器结构　累加器结构利用微处理器中的一个存储单元。该单元称为累加器，设计为算术操作源和算术操作目的单元。仅第二操作数地址作为指令代码的一部分。在算术操作前，第一操作数由载入指令从存储器复制到累加器中。接着运算结果被重新从累加器复制到存储单元中。

存储器/寄存器结构　存储器/寄存器结构由微处理器中一连串寄存器构成。两个操作数都可以由指令编码。然而，只有一个操作数能够根据存储地址直接寻址。另一个操作数和运算结果被存储于其中一个寄存器。在进行运算操作前，其中一个操作数必须要从存储器复制到一个寄存器中，这在某种程度上类似于累加器结构。操作结束后，运算结果需要写回存储器。然而，如果寄存器数量相当多，中间结果就临时存储于这些寄存器中，根据需要来回复制。同样的，“存储器/寄存器结构”是根据操作数存储位置给出的。

载入/存储结构　载入/存储结构每次操作都将两个操作数编址到寄存器中。因此，每次操作前都要先将操作数载入寄存器，运算结果再复制回存储器中。

操作数地址　另一个区别是隐式编码地址和显式编码地址所占位数。为了理解清楚，举一个简单的例子。$C = A + B$ 操作需要三个地址：

（1）操作数 A；

（2）操作数 *B*；

（3）结果操作数 *C*。

显式地址 指令系统结构允许随机选择上面提到的三个操作数地址（也就是提供三个地址显式编码），因此被称为非破坏性指令系统结构。

隐式地址 由于在编码指令中三个地址经常占用很多位数，因此在一些结构中使用隐式地址。在隐式编码中，两个源操作数地址中的一个地址同时作为目的地址。因此，操作结果存储于其中一个源操作数地址中，造成操作数改写（被破坏）。这一破坏过程被称为破坏性指令系统结构。

一个给定微处理器的完整补码指令称为指令系统。除了操作数存储器和操作数地址之间的区别外，微处理器的管理指令系统结构在其他一些方面存在不同，如指令执行大小和（或）指令执行方式[39]。

在面向硬件的编程中，必须考虑到微控制器配置的一些附加细节（其中一些非常特殊）。这包括，例如中断处理、存储器组织、Flash 编程及各种可能的微控制器操作状态（功率衰减模式）。关于这些问题的详细讨论已经超出了本书范围，相关内容请参阅有关的微控制器资料文献。

2.3.3.5 I/O 模块结构

输入/输出模块通过输出信号提供外部信号和控制变量响应。这样，I/O 模块包含微处理器与其运行环境间的连接关系。除了与微控制器内部数据总线连接外，每一个 I/O 模块还设有与外部连接的引脚，例如与传感器和执行器连接。

图 2-15 给出了一个 I/O 模块的原理图[39]。其主要任务可以分为以下几种。

（1）与微控制器内部数据总线通信；

（2）与外部环境通信；

（3）数据保存；

（4）看门狗功能与定时器控制；

（5）故障识别。

1. 寻址

输入/输出模块可以分为以下几种类型。

独立 I/O 两个不同地址区域分别作为微处理器存储器和 I/O 模块存储器。因为 I/O 模块指令比较特殊，所以其编程具有严格的限制。

存储映像 I/O 微处理器和 I/O 模块在通用地址范围内共用一块存储区域。这种安排方式在 I/O 模块中有利于允许大量微处理器寻址指令。但存在一个缺点——占用地址空间，尤其是 4 位或 8 位微处理器。因此，相应于存储映像 I/O 结构，目前微处理器都是 16 位或 32 位字长。

2. 操作模式

另一个区别 I/O 模块不同特征的是可支持操作模式。可以定义为 4 种不同的操作模式。

程序 I/O I/O 模块由微处理器直接控制，微处理器通过一个简单程序处理其全部功能设置。当 I/O 模块执行操作时，微处理器必须等待。因此，这一操作模式用于微处理器只

处理输入/输出任务的情况下（例如智能传感器和执行器的控制）。

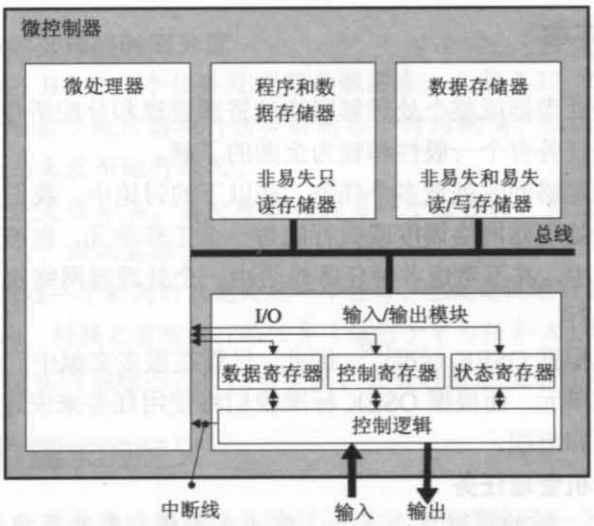


图 2-15　输入/输出模块的典型体系结构

查询 I/O　I/O 模块能够执行独立操作，这期间输入/输出数据被存入特殊缓冲器中。微处理器循环检查 I/O 模块状态及时根据需要传送新的数据。这种操作模式主要适用于基于软件中断系统的微处理器，称为软件中断系统。

中断驱动 I/O　I/O 模块独立执行所有输入/输出操作。通过中断信道，I/O 模块通知微处理器新数据的出现或请求微处理器操作。这一操作模式的根本优势，在于微处理器和 I/O 模块能够进行并行操作。只有在 I/O 模块请求微处理器协助的情况下，微处理器程序才会中断执行。

直接存储器 I/O 存取（DMA）　在这种操作模式中，I/O 模块能够和存储区域进行直接的数据交换而无须微处理器的参与。这一操作模式主要支持高端执行领域微处理器。作为中断驱动 I/O，该操作模式要求硬件排好所有等待请求优先次序，甚至必要时阻塞这些等待请求。

一般地，微控制器 I/O 模块描述面向硬件方面的软件部分集成于软件平台的一层，称为硬件抽取层。如第 1 章图 1-22 所示，本书中省去了硬件抽取层，这些是 I/O 单元与其他系统进行通信所需要的（例如通过数据总线）。通信所要求的软件部分需要单独考虑，其结构在 2.5 节和 2.6 节有所讨论。同样，评估微控制器实时操作响应，对微控制器中断系统的根本理解是基本前提。

2.4　实时系统

一般认为微处理器控制功能的执行（涉及以下章节所提到的处理器）也服从基于时间的要求，所以称为实时系统。本节大致讨论实时系统必要条件、基本原理和结构及几种

实时操作系统。

2.4.1 定义任务

在试图描述单个处理器或整个处理器网络的资源管理和分配所使用的各种方法前，应该对所要处理的所有任务有个一般性和较为全面的了解。

一个处理器网络能够同时处理多个任务。在以下的讨论中，我们所称的任务将用来描述一个处理器或一个处理器网络调度或执行的每一个工作单元，也被称为潜在的或实际的并行处理过程。本文中，并不考虑各种任务是否由一个处理器网络执行还是由一个单一处理器以拟并行方式执行。

本书使用的定义根据 OSEK 标准[16]。因此，尽管在很多文献中广泛使用进程来描述按并行方式处理的工作单元，而根据 OSEK 标准我们将使用任务来表述。本部分讨论与时间有关的处理任务定义和组织。

举例：各种发动机管理任务

在发动机管理中，例如获取单个点火、喷油或踏板参数采集功能任务必须由发动机 ECU 微控制器在预先定义的特定时间内执行完成。在接下来的部分，任务由与时间轴相关的水平线条来表示，如图 2-16 所示。

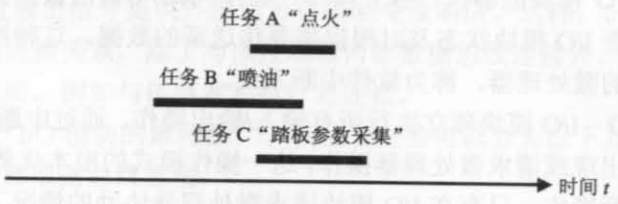


图 2-16 发动机 ECU 处理的任务

为了避免在给定任务（如点火、注油和采集）的处理过程中使用各种不同的名称，我们统称这些为任务执行。

处理器以一定的序列顺序执行一个任务。换句话说，处理器一条接着一条执行指令。在图 2-17 中，一个任务指令执行循序通过时间轴或时间线来表示，遵循从左至右的循序。

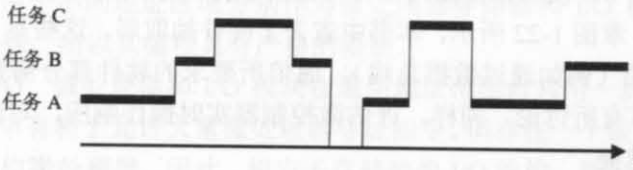


图 2-17 针对任务 A、B、C 的处理器仲裁图表

在一个处理器以拟并行方式执行多个任务的情况下，处理器必须为各任务之间分配一定的时间间隔。在时间线上的特定点处要求两个任务间进行转换。这种基于时间为各任务分配时间间隔的图表称为仲裁图表（图 2-17）。

所谓的仲裁，既表示根据处理器把时间分成一段一段，又表示总线访问允许分配，也

可以称为如 2.4.4 节所介绍的调度，尽管使用后者更加符合 OSEK 标准，尤其涉及到处理器时间分配。

举例：三个任务的处理器时间分配

图 2-17 给出了 A、B、C 三个任务处理器仲裁图表。在图 2-17 中，一个任务可能存在多个不同状态。该图给出了处理器执行任务时的各个时间间隔。根据 OSEK 标准，本书后面将使用这种任务状态来表示运行状况。

图中运行状态首先是任务 A。当处理器转到另一个任务时（图 2-17 中为任务 B），该任务成为当前运行状态，以此类推。

然而，因为处理器在一个时间内只能处理一个任务，也就是说运行状态在任何时间只能表示一个执行任务。因此，转换之前所执行的任务（该例子中为任务 A）要求进入另一种状态。

接下来的部分讨论实时系统的各种任务状态、任务转换触发事件及任务转换不同机制。

2.4.2　实时性需求的定义

首先，对于怎样合理准确地描述任务的时间要求，必须给出一个准确的定义，为此，下面必须明确区分时间点（瞬时）和时间周期（时间间隔）。

2.4.2.1　任务激活和任务最后期限瞬时点

区分给定任务为实时系统还是非实时系统的两个重要参数是任务激活瞬时点和任务最后期限瞬时点（图 2-18）[42]。

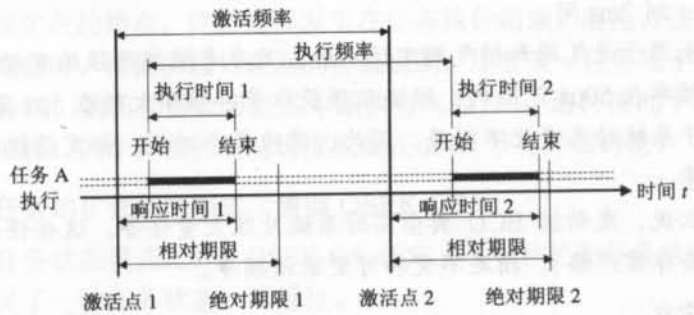


图 2-18　实时性要求的确定

（1）在一个实时系统中，任务激活点就是任务触发或使能瞬时点；

（2）任务最后期限点就是任务执行结束那个时间点；

（3）响应时间是激活点和任务执行结束这一段时间；

（4）给定任务最大允许响应时间也被称为相对期限，任务期限也被称为绝对期限，在时间线上可以通过激活点和相对期限来计算出绝对期限；

（5）给定任务相连两个激活点时间间隔被称为激活频率，一定不要把激活频率和两任务执行时间间隔即执行频率相混淆。

这种任务限制条件利用了任务指定时间窗执行，称为实时需求。在最简单的情况下，一个任务的实时需求可以由它的相连激活点和对应的相对或绝对期限来表示。任务实时需

求经常由激活频率或一个激活事件和一个相对期限来定义。

区分影响任务实时需求（尤其是任务执行时间窗）和任务执行所需时间间隔也称为任务执行时间是非常重要的。如图 2-18 所示，一个任务执行没有被中断时，其执行时间就表示为任务执行开始与结束之间时间间隔。如果存在中断时，任务执行时间就为任务执行开始与结束之间及期间处理器处理中断任务时间之和。

2.4.2.2 硬实时需求和软实时需求

迄今为止的讨论已经确定了对于给定的输入值，实时系统必须要在指定的时间间隔内提供准确的输出值。

实时需求常被划分为两种类型，即硬实时需求和软实时需求。相关文献给出了几种关于硬实时需求和软实时需求的定义。本书中参考了文献资料[42]中的定义规范。

给定任务实时需求可以有硬的方面，如果要求任务在明确指定的时间间隔内必须要完成，该任务就称为硬实时任务。这种情况下获得的确认信息既准确又可靠。另一种类型是，如果没有该种在指定的时间间隔内必须要完成的要求，给定任务所有实时需求将被认为是软的方面，且该任务被称为一个软实时任务。

因此，任务硬实时需求不应与安全性或者任务执行“速度”相混淆。

举例：发动机 ECU 功能实时需求

很多受控发动机子系统具有广泛的动态变化。发动机 ECU 功能实时需求也相应改变。那些功能必须根据特定曲轴位置同步执行，因此要求具有可变采样频率，从而获得最高或最快发动机速度。根据汽缸数量和发动机速度最大值，可以算出喷油和点火最快采样频率范围大约在 1ms 到 2ms 间。

高速采样频率也用于进气阀和排气阀定位功能，及参考燃烧压强的发动机管理功能。阀门定位典型采样频率在 50μs 范围内，燃烧压强获取采样频率大约在 5μs 范围内。

相对地，其他子系统动态变化不明显。因此，其他各个功能，如发动机冷却控制可以采用较低的采样频率。

因此，可以这么说，发动机 ECU 典型实时系统处理大量任务。这些任务要符合多种实时需求（其中一些非常严格），指定不变和可变激活频率。

2.4.2.3 进程定义

当处理具有同样实时需求的不同独立任务时，这些任务可能作为一个任务集来处理也可能集合为一个单一任务来处理。本书将通篇使用进程这一概念。一连串具有相同实时需求的进程可以集合为一个单一的任务（图 2-19）。这里，并没有指定各种进程的实时需求而是对于由此得到的任务给出实时需求。给定任务进程按照指定顺序执行。

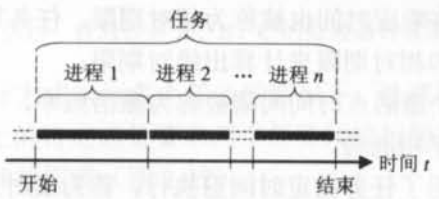


图 2-19 进程和任务的确定

2.4.3　任务的状态

2.4.3.1　任务的基本状态模式（遵循 OSEK-OS）

如图 2-18 所示，依照任务实时需求并不能强制激活点刚好是任务实际执行开端（也就是起始点）。在激活点和执行间隔期间，任务进入一种特殊的中间状态，例如假定处理器这时正在执行另一个任务。根据 OSEK，本书称这种状态为准备状态。相对地，任务进入激活前和执行后状态时称为悬挂状态。这些状态及其之间相互转换过程可以通过状态机来形象地表示。基于 OSEK-OS 标准的基本任务状态模式如图 2-20 所示（OS 为 Operating System 的缩写）。

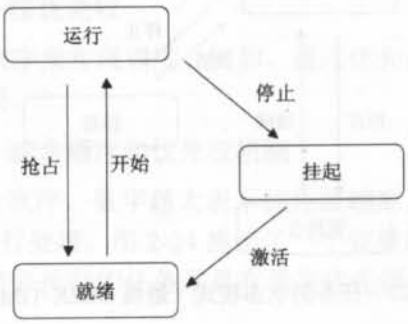


图 2-20　任务的基本状态模式（遵循 OSEK-OS）

任务转换过程包括激活、开始、抢占和终止。如图 2-18 所示，激活过程发生在激活点，开始过程发生在起始点，终止过程发生在任务执行结束。而抢占过程发生在多个任务竞争同一个处理器时。根据所选处理器的仲裁机制，尽管某一任务处于运行状态，当出现优先级较高任务时，当前任务就可能被中断执行（抢占）。这种情况下，被抢占任务进入抢占任务转换过程。关于仲裁机制或调度机制在 2.4.4 节有详细讨论。

2.4.3.2　任务的扩充状态模式（遵循 OSEK-OS）

除了基本任务状态模式之外，OSEK-OS 还定义了一种扩充任务状态模式。如图 2-21 所示，另外定义了一种任务状态，即等待。

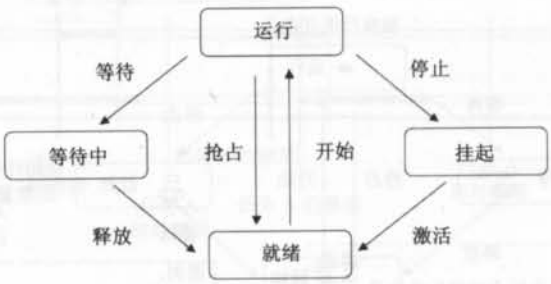


图 2-21　任务的扩充状态模式（遵循 OSEK-OS）

在特定情况下，某一任务可能需要中断执行以等待允许其继续执行的事件。这时任务进入等待状态直到重新执行。进入等待状态转换过程由任务自身触发。在等待状态期间，

处理器处理另一个任务。扩充状态后相应转换过程称为等待和释放。

2.4.3.3 任务的状态模式（遵循 OSEK-TIME）

为了便于理解时控仲裁机制，按照 OSEK-TIME 定义了时钟触发任务状态模式，如图 2-22 所示（TIME 代表 Time-Triggered Operating System）。关于时钟触发仲裁机制本章后面 2.4.4.6 节有详细讨论。这种任务状态模式分为三种，即悬挂、运行和被抢占状态。悬挂状态向运行状态直接转换的过程称为激活。缺少了准备状态环节意味着任务执行激活点和起始点是同一点。

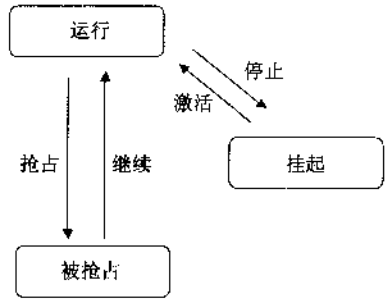


图 2-22 任务的状态模式（遵循 OSEK-TIME）

按照 OSEK-OS 标准任务状态模式，当前执行任务能够被其他任务中断执行。这种情况下，被中断任务进入被抢占状态。

2.4.4 处理器的调度策略

本部分讨论几种处理器调度机制。这种机制首先考虑在多个任务竞争处理器时间时如何做出选择。利用 OSEK-OS 扩充任务状态模式，图 2-23 给出了一个相似的例子。五个任务同时处在准备状态竞争处理器时间。

一般来说，这种情况下一些任务会进入各自可能的状态。分别是非激活任务组处于悬挂状态、准备任务组处于准备状态、等待任务组处于等待状态及执行任务组处于运行状态。当然，在单个处理器的情况下，执行任务组只能包括一个任务[43]。

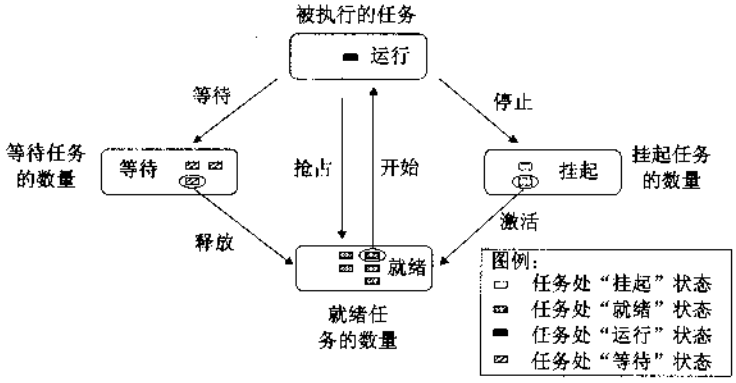


图 2-23 通过不同的状态来管理任务

除了本部分讨论过的几种任务状态模式，基于 OSEK 实时操作系统也支持多种处理器调度机制。调度机制执行所需的操作系统组成部分称为调度程序。启动执行所需的组成部分称为调度器。实时操作系统结构将会在本章后面的 2.4.5 节中进行讨论。

2.4.4.1　处理器调度——按次序顺序

针对一组处于准备状态的任务可以采用任务激活顺序次序进行处理器分配的调度机制。这种调度是根据 FIFO（先入先出）原则对准备状态任务组进行队列排序的。

简单来说，这意味着后来激活任务需要等待前面已激活任务执行结束后才进入执行状态。显而易见，这种调度机制在某些情况下占用较多时间。

2.4.4.2　处理器调度——按优先权

也可以不根据任务激活次序来实现调度，例如，通过优先权等级制定调度规则，根据优先权高低将准备任务组分类。

2.4.4.3　处理器调度——结合顺序和优先权机制

OSEK 标准任务评定优先次序，数字越大表示优先级越高。当任务具有相同的评定优先级时，则根据 FIFO 原则进行处理。图 2-24 描绘了一个完整的基于结合机制的准备任务组调度管理。因而，下一个将要执行的任务是具有最高优先级的“长者”，如图 2-24 中左上角所示。

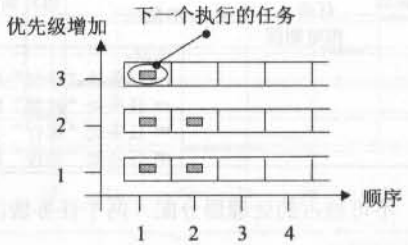


图 2-24　处在准备状态的任务组管理（遵循 OSEK-OS）

因此，仲裁图表模轴经常以优先级上升次序排列任务，如图 2-25 所示。

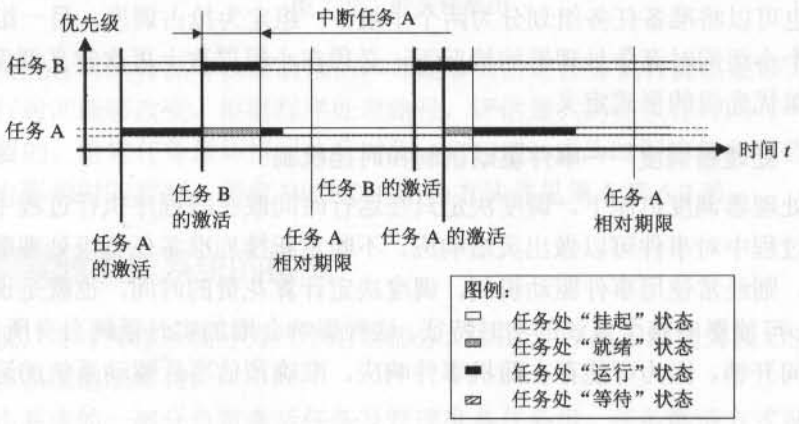


图 2-25　可抢占的处理器分配（两个任务情况）

另一种优先级控制仲裁机制区分标准是高优先级任务调度时是否发生任务执行抢占及哪一个任务处于运行状态。因此，可以分为抢占调度和非抢占调度。

2.4.4.4 处理器调度——抢占机制

抢占处理器调度机制允许高优先级任务中断低优先级任务执行。如果该中断发生在任务执行的任何一点，则称为全抢占调度，如图 2-25 所示。较高优先级任务 B 一进入准备状态，任务 A 的执行就被中断。任务 A 只有在任务 B 执行结束后才能继续执行。

2.4.4.5 处理器调度——非抢占机制

非抢占处理器调度机制规定从低优先级任务到高优先级任务的转换只发生在几个特定时间点上。例如，发生在当前低优先级任务执行结束后。或者是，转换发生在低优先级任务集合成的所有过程执行结束后。这样导致了低优先级过程或任务不能被中断，并延迟了高优先级任务的执行。任务 A 和任务 B 的这种调度方案如图 2-26 所示。

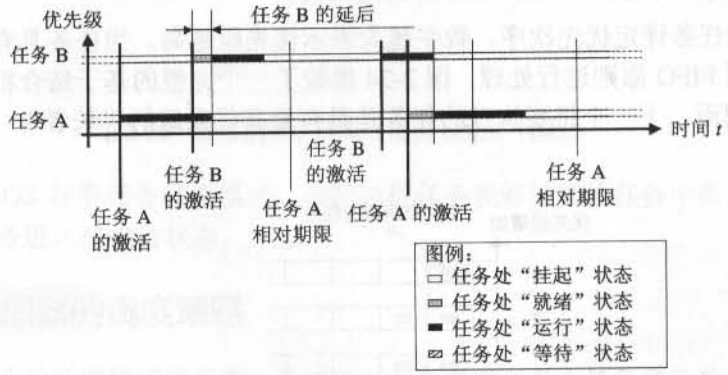


图 2-26 不可抢占的处理器分配（两个任务情况）

在实时需求和任务状态中，优先级是每个任务都具有的属性。

然而，抢占调度和非抢占调度之间的区别并不是任务属性而是调度机制。这种机制应用于必须调度的整个准备任务组。

例如，也可以将准备任务组划分为两个小组，一组定为抢占调度，另一组定为非抢占调度。在两个小组同时竞争处理器的情况下，必须在小组层次上再次定义调度机制——可能以各个小组优先级的形式定义。

2.4.4.6 处理器调度——事件驱动机制和时控机制

在动态处理器调度机制下，调度决定只在运行时间或在线程序执行过程中做出。这意味着在执行过程中对事件可以做出灵活响应，不断重新排列准备任务组处理顺序。如果在这种情况下，则经常使用事件驱动机制。调度决定计算花费的时间，也就是说，调度器自身执行时间，可能影响整个系统的实时特征。这种影响会增加实时系统自身所需执行时间，称为运行时间开销。因为可能存在随机事件响应，准确预估事件驱动系统的运行时间特征是不可能的。

相对地，静态处理器调度机制允许所有调度决定在程序执行前或离线时做出。因为这

种机制要求预先知道所有事件，显然这种方法存在事件响应限制。因此，响应都是预先定义的，所以称为时间依赖事件。因此经常采用时间驱动机制。对整个系统实时特征中调度决定计算花费的时间带来的影响是微乎其微的，因为只需要调度器一个比较简单的功能。实时操作系统运行时间开销相对地也就比较低。

在实际执行前我们可以计算出仲裁图表并绘出调度表。这样，某一特定时间点定义为任务激活点。调度功能根据调度表在预先确定的瞬时点开始任务执行。

图 2-27 给出了一个调度表实例。相应的仲裁图如图 2-28 所示。这里所描绘的每一个任务都定义了一个固定的时间窗。当以表中调度序列处理时，从表项目中第一个任务开始执行直至最后，并不断重复执行该调度序列。本例中，一个调度循环为 40 个时间单位。

时刻	动作	
0	任务 A 开始	调整周期
8	任务 B 开始	
11	任务 C 开始	
20	任务 A 开始	
31	任务 D 开始	
35	任务 E 开始	

图 2-27　静态调度表

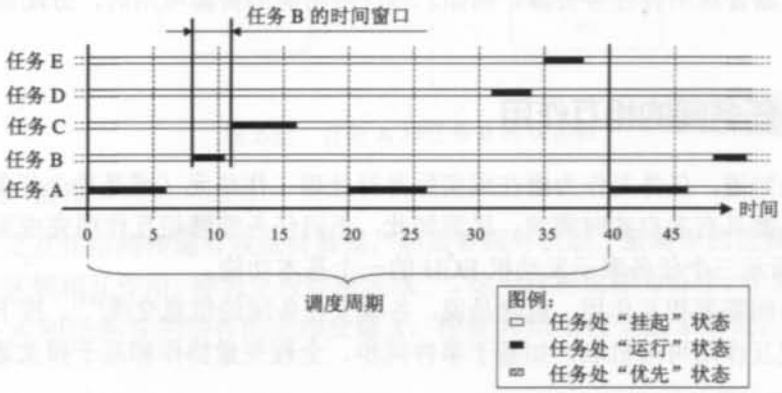


图 2-28　静态仲裁图

这种系统运行时间特征可以准确预估，并能够为给定任务执行提供足够大的时间窗。因为任务执行时间能够改变，根据程序处理路径，评估最大所需执行时间（一种最坏情况评估）是必要的。给定任务最坏情况执行时间（WCET）成为时间窗低门限值定义的决定变量，同时也影响时间控制。确定 WCET 的适当方法详见第 5 章 5.2 节。

2.4.5　实时操作系统的构成

一般来说，实时操作系统分为三个主要部分。如图 2-29 所示结构为按 OSEK-OS 所描绘的实时操作系统组织结构。

（1）操作系统的一部分负责激活任务及管理准备任务组。任务激活方式可以为时间倚赖（基于实时时钟）也可以为事件驱动（例如中断）。实现这一功能，该部分需要所有任

务激活点和激活事件相关信息。

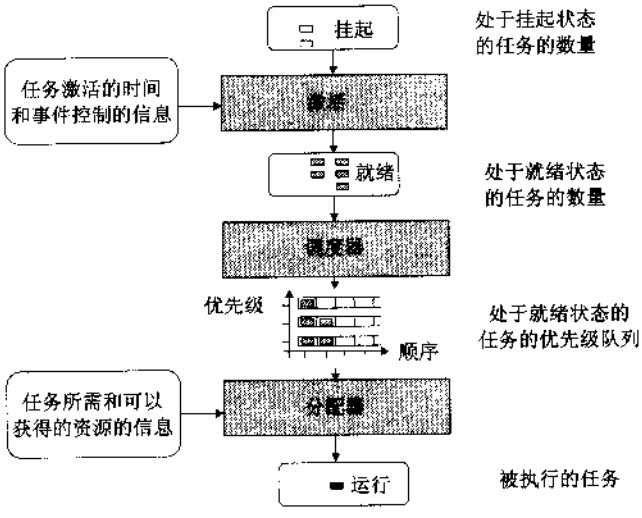


图 2-29 实时操作系统组织结构简图

（2）调度器评估准备任务组并根据处理器调度机制设定执行次序。

（3）分配器管理所有任务资源。例如，当任务所需的资源可用时，分配器就执行最高优先级任务。

2.4.6 任务间的相互作用

我们已经知道，任务是作为潜在或实际并行处理工作单元（或是单个任务）引入的，所有这些任务都具有各自实时需求。尽管如此，不同任务能够相互作用完成某一功能。例如，图 2-16 所示三个任务表示发动机 ECU 的一个基本功能。

因此任务间需要相互作用，也就是说，各单个任务间的信息交流[43]。接下来的部分讨论内部任务交互作用可用机制，如基于事件同步、全程变量协作和基于报文通信。

2.4.6.1 同步

图 2-30 给出了基于事件的两个任务间相互作用的处理顺序。这种类型图表称为报文序列图表[44]。以下部分都将用报文序列图表来描述任务交互作用。报文序列图表中时间轴 t 的方向会从上到下。

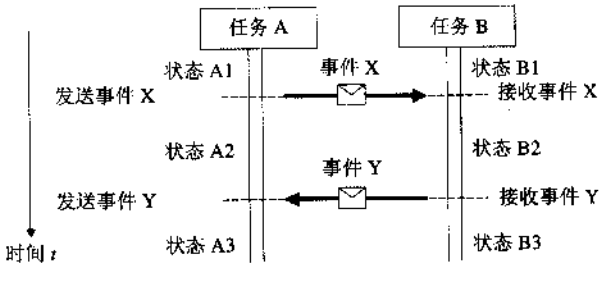


图 2-30 描述两个并发任务 A 和 B 同步的信息序列图

在本例中，任务 B 接收事件 X，致使任务 B 从状态 B1 转换到状态 B2。接着，相应反馈即向任务 A 发送事件 Y 致使任务 A 进入相应的状态转换。这样，拟并发（或拟并行）任务同步就完成了，确保了逻辑处理顺序。只有在接收事件 X 后也就是在状态 A1 后任务 B 才进入 B2 部分或更准确的说 B2 状态。只有在接收事件 Y 后也就是在状态 B2 后任务 A 才进入 A3 状态。

因为任务是拟并行执行的，所以可能出现冲突（例如，多个任务试图存取共用资源时）。这里描绘了几种典型冲突的例子（参见图 2-33 和图 2-35）。任务间的交互作用机制都必须要考虑这冲突的可能性，并运用同步机制来解决冲突问题。

分布式系统中所有的状态组和引起状态转换的事件组都可以由状态机来描述（图 2-31）。

这种类型任务互动只适合单一事件且没有内容信息（也就是数据传送），称为同步。实时操作系统一般支持多种内部任务同步机制。

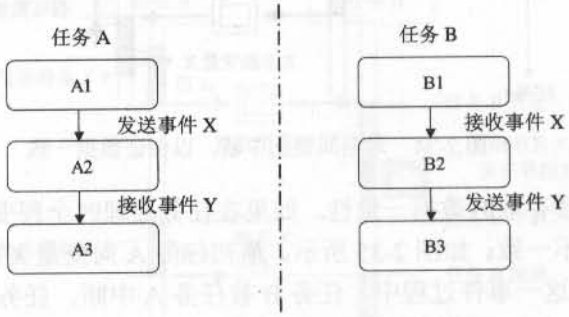


图 2-31　任务 A 和任务 B 的状态机

2.4.6.2　协作

如果在交互作用间传递有效载荷数据，则需要额外机制。最简单的选择是不同任务通过共享数据区域相互作用，称为全程数据区域[43]。这一过程也称为协作。作为例子，图 2-32 给出了任务 A 和任务 B 间协作的全程变量 *X*。任务 A 给全程变量 *X* 赋值，任务 B 读取该变量值。

我们注意到这种使用全程变量的方法可能在一定情况下导致实时系统数据不一致。图 2-33 描绘了这种主要类型。任务 A 在对全程变量 *X* 进行写操作时，被任务 B 中断。因为写存取还没有结束，这时变量 *X* 包含无效或不一致数据片断，而这时任务 B 对其进行了读取。因此，对这一不一致数据片断的进一步处理可能会导致完成无法预知的结果。

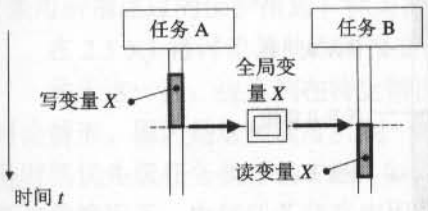


图 2-32　应用全局变量的协作

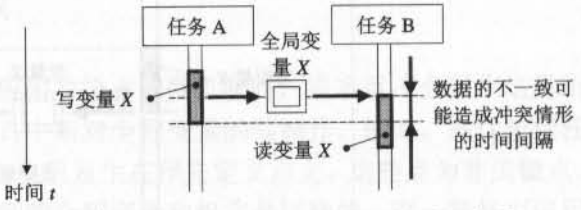


图 2-33　不一致的数据和全局变量

因此，针对任何协作机制都要确保前后数据的一致性。鉴于其重要性，这一要求在许

多特定条件都有明确表述：在任务 T_i 开始与结束点之间必须确保数据的准确。任务 T_i 所有存取数据能且只能由任务 T_i 本身来改变。

有两种方法可以达到这一要求。

方法一 在写存取时确保数据一致。在对全程变量 X 进行写存取期间锁住所有中断请求（图 2-34）。而在原子操作（Atomic Operation）情况下并不要求如此，也就是说，操作由处理器以连续方式处理。例如，对于 16 位字长处理器来说，对 8 位和 16 位变量写操作称为原操作，与对 32 位变量写操作是不同的。因此，只有当对高于 16 位变量进行写操作时，16 位字长处理器中断限制才会激活（例如 32 位变量写操作）。

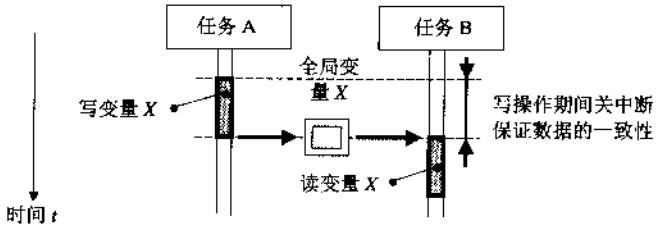


图 2-34 写期间禁止中断，以保证数据一致

方法二 涉及读存取时数据一致性。如果在任务处理时全程变量被反复读取，结果可能与任务执行变量不一致。如图 2-35 所示。最初任务 A 向变量 X 赋值 x_1。接着任务 B 读取处理该 x_1 值。在这一事件过程中，任务 B 被任务 A 中断，任务 A 再次对变量 X 赋值，而这次赋值为 x_2。当处理继续执行时，如果任务 B 使用 x_2 进行计算，就不能确保其输出变量的一致性，导致无法预知的结果。

例如对除法任务 B 即 $Z = Y/X$ 中 X 的进一步处理。尽管除法只有在 $X \neq 0$ 时才有结果，但还是有可能出现 $X = 0$ 的情况。例如，发生在完成除法操作前，对 x_1 是否等于 0 的检验过程中，及除法运算期间任务 B 在检验和除法操作时间间隔内被中断而使用 $x_2 = 0$ 的值。

读存取期间必须通过基于定义事件同步确保数据一致性。

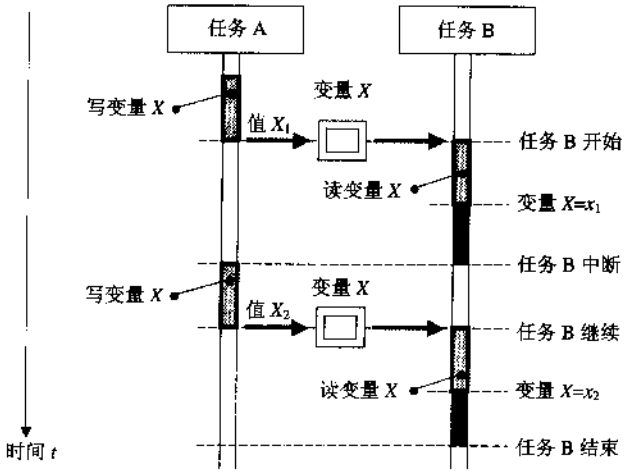


图 2-35 任务 B 被任务 A 中断而导致数据不一致

2.4.6.3　通信

任务间通过局部数据区域相互作用时都要求数据传送。一种处理数据传送机制称为通信[43]。这种通信方法也可能用于解决图 2-35 所描述的同步问题。这样，任务执行时全程变量读写存取的数据一致性就能通过额外的复制机制来确保。

对于读存取变量，这意味着每次任务开始执行时都会进行输入变量局部复制——不用管在哪个时间点需要这些输入变量。也就是说，任务执行起始点成了定义同步点。同步点有效值用于任务整个执行过程。这种复制机制也要求锁存所有中断请求。

对于图 2-35 所示例子，这意味着变量 X 的局部复制在任务 B 开始执行时被存储起来。在下面的图 2-36 图表中，这一复制过程保存了值 x_1 。因此，任务 B 在整个执行过程中都使用该复制过程所保存的 x_1 值。

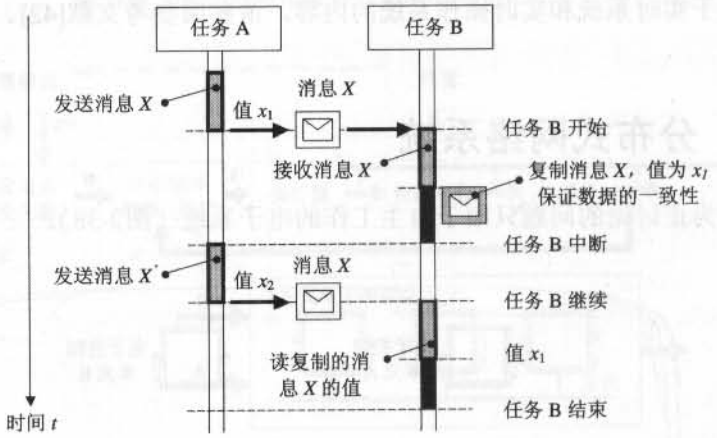


图 2-36　通过局部的信息复制保证数据一致

任务对变量的反复写存取机制也是类似的。同样的，首先进行内部复制，然后只有在任务执行结束后才将复制内容写入全程变量。也就是说，任务执行结束点也可能被定义为同步点。

显然所描绘的机制必须适用于给定任务的所有输入和输出变量。因此，必须确保所有的输入输出变量与其自身相一致，且当变量 Y 为最新值时，不允许变量 X 仍使用原来的值。

本书（根据 OSEK）涉及作为报文机制的内部任务通信机制。因此，使用所谓的报文发送代替全程变量写存取（采用前面所述的保护措施）操作。相反的，从全程变量读存取（采用所描述过的保护措施）称为报文接收。

在 2.5.5.1 将讨论其他不同报文机制。

关于这一点，注意到在特定情况下，应用非抢占调度机制时，根本就不会发生这里所讨论情形。原因是这种调度机制一般不允许中断对全程变量的写操作。同时，高优先级任务对低优先级任务执行的中断（如果可能）也只发生在预先定义点上，这些点为非关键点。在这种情况下，内部任务交互作用所涉及到的全程变量和报文是同样的。这一特征可以用于通过降低多余复制量达到所需存储器和运行时间资源离线最优化。第 5 章 5.4 节对相应方法有更多讨论。

2.4.6.4 逻辑体系结构中任务间的交互作用

前面所介绍的例子清楚地表明，从逻辑层面来描述任务间的交互作用，在许多情况下是有优势的。从逻辑层面来描述任务间的交互作用是对于基于事件、全程变量和报文的现实技术实现的抽象化。接下来从逻辑层面来描述任务间的交互作用将使用报文序列图表，这些不同的任务可能在同一个处理器上执行或在不同的处理器上执行。如图 2-37 所示，描绘了单处理器或多处理器执行的任务交互作用。在这里，机制中所涉及到的事件、全程变量和报文部分，采用信号来代替。因此，在图 2-37 中，任务 A 发送一个信号 X 给任务 B，这可以通过多种机制来实现。

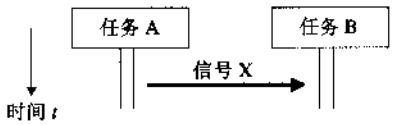


图 2-37 任务 A 和 B 之间交互关系的逻辑示意图

更多关于实时系统和实时操作系统的内容，请参阅参考文献[42]、[43]、[45]及 OSEK 规范。

2.5 分布式网络系统

到现在为止讨论的问题只限于自主工作的电子系统（图2-38）。

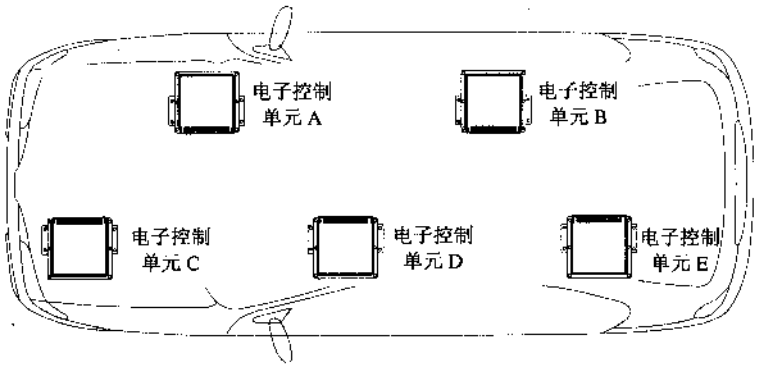


图 2-38 电子系统间的独立运行

对电子系统功能要求的增加，在较早以前就提出一个新的概念，也就是说，从单个自主工作系统向整个集成系统的转变。这种集成化的系统要求掌握汽车中所有的重要功能和信号的相关信息。这种集成系统将车辆中各种 ECU 互连在一起，并有望实现一些跨多个电子控制单元的复杂功能（图 2-39）。驱动力控制系统就为这一类型，发动机和 ABS ECU 协同工作实现高级功能。因而，这种方法不仅要求个体最优化，而且要求促进整体最优化。

然而，正如我们所料，分布式网络系统的设计和实现面临多种挑战。除了在单个处理器上实现拟并发处理外，多个相互依赖的任务在多个处理器的空间分布及真并行执行（称为真并发任务）交互作用已经发展成为主要实现任务。各任务交互作用由通信网络处理（图 2-40）。

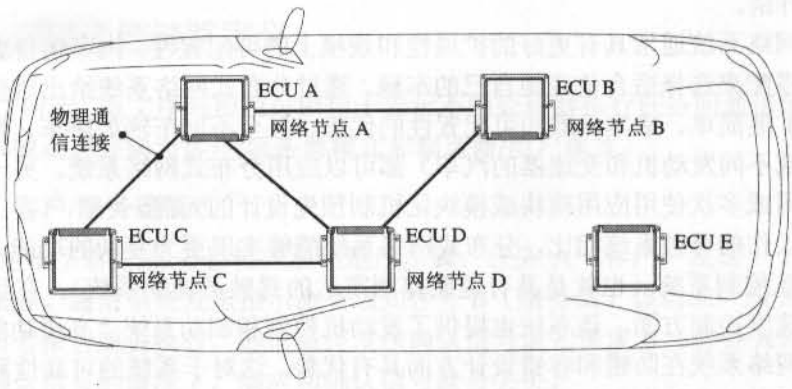


图 2-39　ECU 网络化构成一个分布式网络系统

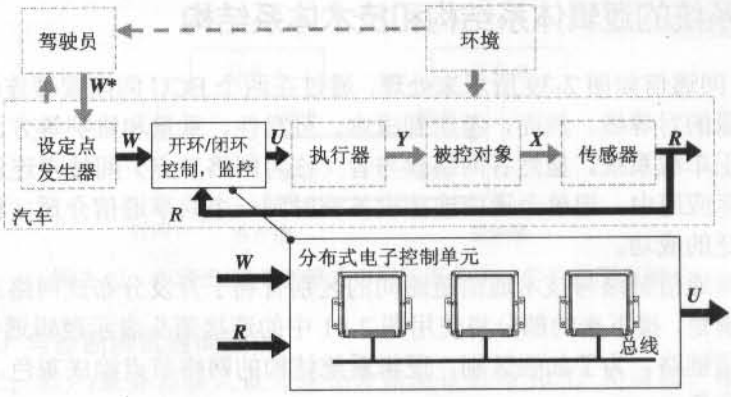


图 2-40　通过 ECU 网络实现控制和监视功能

分布式系统的一些特征、性质及系统机制在 2.4 节都有所提及。为了更好地理解接下来的部分，下面进一步介绍有关的概念。

本书中，对于分布式网络系统的定义采用参考文献[38]中所述定义：

一个分布式网络系统由多个相互间通信的子系统构成。在一个分布式网络控制系统中，数据控制、硬件和数据本身（至少一部分）在系统中分散地组织。

一个分布式网络系统常常由多个自带存储器的处理器构成。处理器通过通信网络相互连接。系统在不同局部区域以并行方式进行控制并处理拟并发任务控制。处理数据分布于不同存储区域。因此，这种配置于车辆中的电控单元网络（或 ECU 网络）称为分布式网络系统。

与集中式系统相比，分布式网络系统应用于汽车设计方面拥有很多优势。

完成某个功能的系统可以在空间上分布，这些分布的电子控制单元共同来完成一个功能。例如车体系统的车门控制系统，在很大的空间范围内分布。车体系统的各单独系统，包括车门上各种系统（例如车锁系统、电动车窗单元和后视镜调节器），顶棚区域各系统（例如遮阳板滑动控制或软车顶控制器），行李箱后挡板控制器和座舱内部系统（例如座椅调节器和转向盘调节器）等，必须相互协作。与集中式系统相比，分布式网络系统减少了在系统

布线方面的开销。

分布式网络系统通常具有更好的扩展性和规模上的可配置性。汽车消费者能够通过可选附加功能搭配来选择适合他或她自己的车辆。通过分布式网络系统给出这些可选功能模块，则具有扩展简单、成本低廉和可配置性的优势。甚至不同车辆如轿车、敞篷车、客车和旅行车（或不同发动机和变速器的汽车）都可以应用分布式网络系统。另一个好处就是可以重复使用或多次使用应用模块或模块化机制预先设计的元件。

与自主工作的单独系统相比，分布式网络系统能够实现更为复杂的功能。这方面例子有自适应巡航控制系统（也就是具有巡航探测雷达的驾驶员辅助系统）。在适应交通运输流量的车辆速度控制方面，该系统也提供了发动机控制和制动系统“高级功能”。

分布式网络系统在防错和容错设计方面具有优势。这对于系统的可靠性和安全性方面具有重要作用。本章第 2.6 节对可靠性和安全性方面有详细讨论。

2.5.1 系统的逻辑体系结构和技术体系结构

如果 ECU 间通信如图 2-39 所示来处理，通过在两个 ECU 间配置硬连线通信链路，则会导致使用大量的对等线。然而，考虑到成本、可靠性、重量和维护等方面因素，这种方法就不能应用于车载系统。显然各网络参与者（称为网络节点）间技术连通性要求能够简单实现。在实际应用中，把单个通信连接设备映射到一个共享通信介质（称为总线）的方法，获得了广泛的成功。

因此，逻辑通信链路与技术通信链路间的区别有利于开发分布式网络系统。

为了表示清楚，接下来的部分将使用图 2-41 中的连接箭头表示逻辑通信链路，使用实线表示技术通信链路。为了加强区别，逻辑系统结构的网络节点涂成灰色，而技术系统结构网络节点为白色。

由网络节点间逻辑通信连接和技术通信连接（也就是共享通信介质，即总线）组成的分布式网络的设计、试运行和测试面临诸多挑战。

因为可能存在多个网络节点同时竞争总线访问权，所以通信系统必须确保任何时刻总线上只有一个网络节点发送数据。多种总线仲裁机制（称为总线访问）将在本章的 2.5.6 节中讨论。

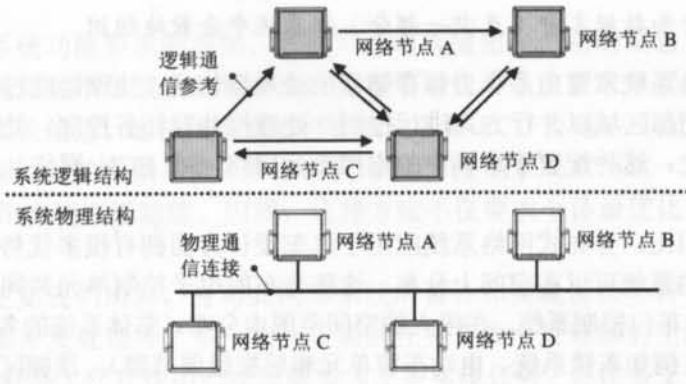


图 2-41 分布式和网络化系统的逻辑和技术体系结构

2.5.2　逻辑通信链路定义

如图 2-37 所示，报文序列图表可以用于表示不同处理器执行任务间通信链路。通信链路有两种基本模式，分别为客户/服务器模式和制造商/用户模式。

2.5.2.1　客户/服务器模式

图 2-42 给出了客户/服务器模式通信顺序步骤。任务 A（客户）发送一个请求命令请求通信系统服务。通信系统发送指示命令通知任务 B（服务器）该请求。服务器向通信系统发送响应信号报告服务执行。通信系统发送确认信号通知请求方（任务 A）服务执行。在没有获得确认信号的情况下，响应和确认信号就会丢失。

甚至在存在多个客户或多个服务器的情况下，客户/服务器模式的客户和服务器之间一般为对等关系。

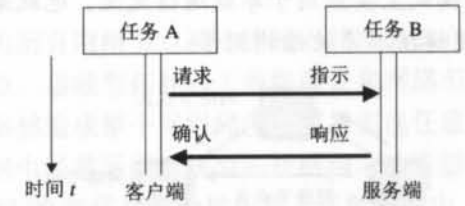


图 2-42　在客户/服务器模式中用于服务确认的信息序列图

举例：ECU 与诊断测试器间通信

车载系统上，客户/服务器模式最为适合诊断测试器与 ECU 间通信。诊断测试器将用户指令转换成与 ECU 间的事件驱动通信。为了达到这一目的，诊断测试器作为一个临时网络节点，扮演客户角色发送标准请求，请求服务器（ECU）服务。图 2-43 给出了一个逻辑技术系统结构例子。

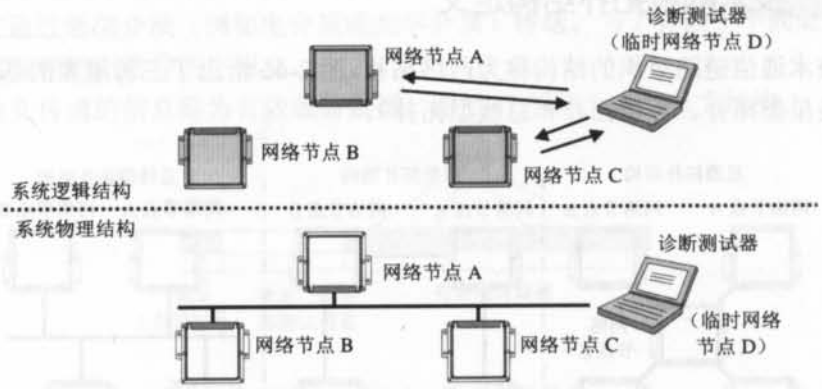


图 2-43　诊断测试器和 ECU 间的非车载通信

2.5.2.2　制造商/用户模式

图 2-44 给出了制造商/用户模式通信顺序步骤。这种逻辑类型表示法适合描绘这样一

种服务，事先提供一个任务（制造商程序），及几个无预先请求信息任务（用户程序）。

制造商/用户模式表示一个制造商程序与多个用户程序间的关系。因此，该模式适合向一组网络节点或向所有网络节点发送信号（广播关系）。

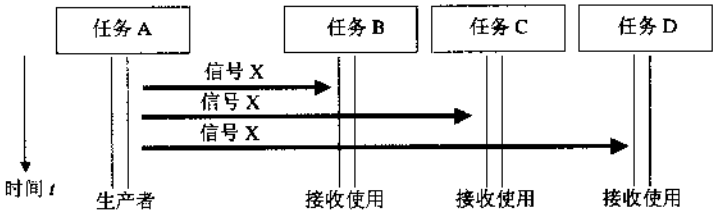


图 2-44 制造器/用户模式中服务的信息序列图

例子：车载 ECU 通信

制造商/用户模式适合于实现由多个网络节点分布实现的控制和监控功能，并具有周期性的信号交流。因此，这种模式主要应用于车载通信定义，也就是，多个网络式车载 ECU 通信。图 2-45 给出了一个逻辑技术系统结构例子。

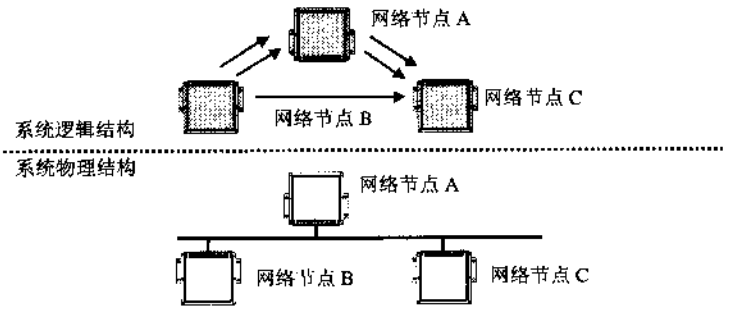


图 2-45 ECU 间的车载通信

2.5.3 技术网络拓扑结构定义

决定技术通信链路组织的结构称为网络拓扑。图 2-46 给出了三种重要的基本结构示意图（也就是星型拓扑、环型拓扑和总线型拓扑）。

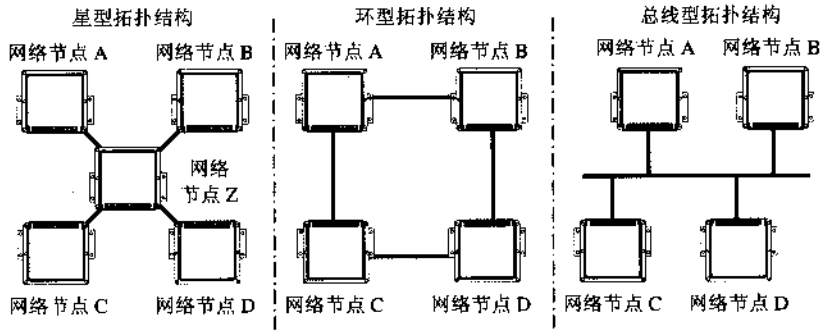


图 2-46 网络拓扑结构

更加复杂的网络拓扑可以通过这三种基本结构为基础进行组合。单独网络区段可以通过网关进行互连。

2.5.3.1　星型拓扑

在星型拓扑中，各网络节点通过对等线与中枢网络节点 Z 互连。所有通信都通过中枢网络节点 Z 来处理。因此，当有 n 个网络节点时，节点 Z 要求有$(n-1)$个接口。中枢节点 Z 的失灵将导致整个通信瘫痪。

2.5.3.2　环型拓扑

环型拓扑为一个闭合的对等线环。所有网络节点都作为能动元件，能够重建和发送入站信息。这种拓扑允许实现大规模空间扩展网络。然而，值得注意的是，除非引入适当的措施（例如故障节点探测和桥接），否则单个网络节点故障可能瘫痪整个网络。

2.5.3.3　总线型拓扑

总线型拓扑的特征为所有网络节点被动连接入共享通信介质。一个网络节点发送的信息可以到达其他所有节点。总线型拓扑易于线缆铺设和网络节点连接，具有简易扩展性。一个网络节点故障不会必然造成整个网络瘫痪，容易实现任意逻辑通信关系。

鉴于这些优势，车辆中经常采用总线型拓扑结构。总线型拓扑典型代表就是控制器局域网（CAN）[2]，20 世纪 90 年代初开始被应用于车辆系统中。

2.5.4　报文定义

在大多数汽车应用中，都使用串行通信系统。这就要求不同处理器执行任务间信号必须以串行方式传递。其通过将传递信号嵌入标准报文帧来实现，多数情况下都定义了报文帧大小。装入信息的报文帧称为报文。

串行传送过程存在一个信号分成多个报文的情形；相反的，一个报文也可以传送多个信号。报文通过通信介质（例如电介质或光学介质）传送。为了有助于不同处理器执行任务同步，可能使用无信号的空报文。

通过报文传递的信息称为有效载荷数据，图 2-47 所示为典型报文结构。

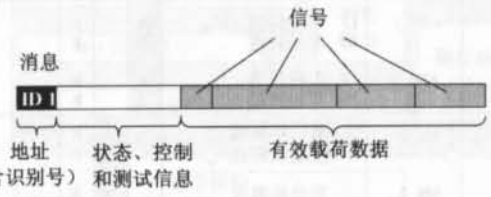


图 2-47　消息、数据和信号

除了有效载荷数据外，报文帧还包含自身属性，如目的寻址标志符及所需的状态、控制和校验和信息，例如用于识别和处理传输误差。

2.5.4.1 寻址

寻址用于映射报文发送方与接收方之间关系。这里，寻址分为节点寻址和报文寻址。

例如，如果一报文从网络节点 A 传输至网络节点 B，节点寻址将网络节点 B 地址写入报文标志符。接收报文时，每个网络节点都将自身地址与该报文标志符相比较，地址相匹配则接收处理。

然而，如果每个报文都有明确的地址，则引入报文就能轻易被多个网络节点接收处理。在这一处理过程中，各网络节点使用帧过滤来确定引入报文是否符合各自节点。这种寻址方式的优点在于给定多个网络节点所需报文只需发送一次，并能同时被所有接收网络节点接收。

2.5.4.2 通信矩阵

一个网络所有通信关系可以整理为发送方/接收方关系表，称为通信矩阵，或简称为 C 矩阵，C 矩阵包含所有通信网络信息。

图 2-48 引用了部分 C 矩阵。左边第一栏列出所有网络节点（也就是所有网络 ECU）。右边接下来两列为报文和各网络节点以信号形式发送的有效载荷数据。剩余一栏再次表示网络节点。这里，报文发送方和接收方由代码字母表示。字母“R”代表接收方，表示报文接收评定网络节点。而字母“S”表示报文发送方。

2.5.5 通信网络管理组织结构

尽管在技术网络层面，是报文进行发送和接收，而在逻辑网络层，在许多的情况下感兴趣的是网络传输的数据（也就是传输信号）。因此，各网络节点需要一个信号与报文间映射关系处理环节。这一通信系统环节称为传输层。图 2-49 所示为一种通信模式概述。

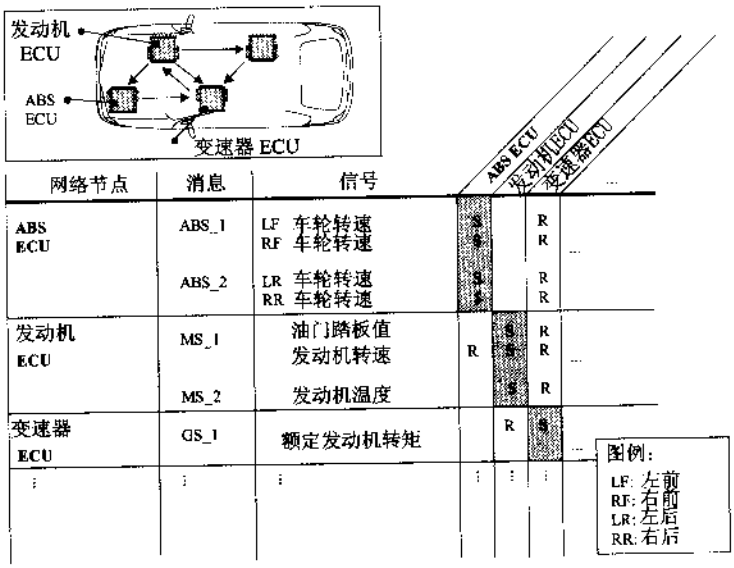


图 2-48 通信矩阵

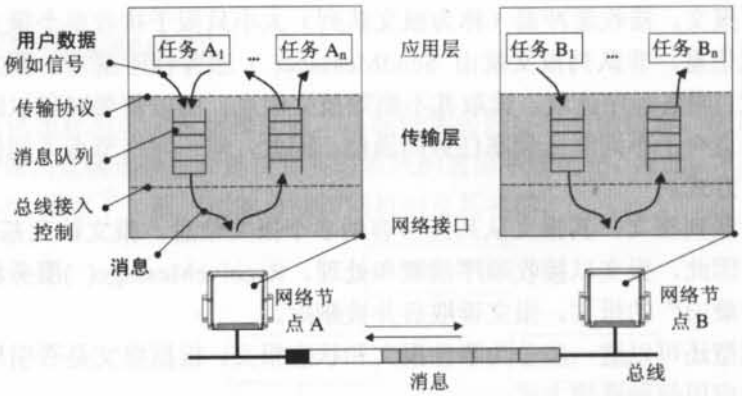


图 2-49　通信模型

本部分详细介绍传输层（遵循 OSEK）结构。接着介绍遵循 ISO 标准的数据通信参考模型，称为开放系统互连参考模型，也称为 OSI 模型。

OSEK-COM（COM 表示 Communications）定义网络节点间通信软件构成及标准化联合接口。OSEK-NM（NM 表示 Network Management）提供网络管理相关定义。图 2-50 所示为软件构成概述。

因为几乎所有车载 ECU 间互连，所以这种全车通信层结构标准化有利于系统的规范化、整体化和质量保证。本文中，OSEK 标准涵盖车载通信领域。而车外通信标准（例如，维修车间诊断支持和软件更新）由 ASAM 和 ISO 标准定义[17, 25, 26]。

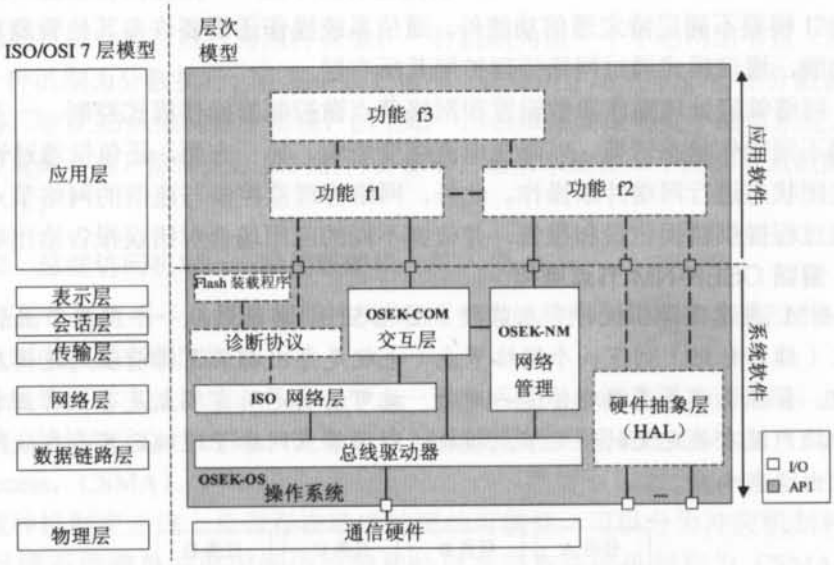


图 2-50　软件组件结构（依据 OSEK-COM V3.0.1）

2.5.5.1　通信（遵循 OSEK-COM）

OSEK-COM 定义了多个报文机制变量，该报文机制在 2.4.6.3 已有所介绍。对于任务间通信传输报文，可以分为队列报文和非队列报文。

对于非队列报文，接收缓冲器（称为报文队列）大小只限于接收单个报文信息。一旦接收到新的报文信息，非队列报文就由 SendMessage() 服务程序重写。报文内容可以由 ReceiveMessage()服务程序读取。读取并不删除报文信息，所以需要时可以随时读取。这种通信方式特别适合于不同激活频率任务间通信。因此，同一网络节点不同任务间通信一般采用这种通信方式。

然而，对于队列报文，其报文队列能够容纳多个报文信息。报文队列基于 FIFO（先入先出）原则。因此，报文以接收顺序读取和处理。ReceiveMessage()服务程序总是从报文队列中读取“最旧”的报文。报文读取后并被删除。

基于报文类型还可以进一步分为事件报文和状态报文。根据报文是否引用事件或状态变量值，来决定应用何种通信方式。

在每个发生事件相关联的情况下，个别报文的丢失可能造成发送方和接收方不同步。这种确保同步的报文称为事件报文。

一种相关应用为使用增量编码器测量转动速度。某一事件的丢失（这里为速度信号传输）造成速度计算错误。

相对地，如果用到状态变量当前值，就可能将原有值改写为当前值。这种情况下用到的报文称为状态报文。

一种相关应用为温度测量。多数情况下，温度传感器单个测量值丢失还是允许的，因为这并不会造成温度计算错误。

2.5.5.2 网络管理（遵循 OSEK-NM）

除了 OSI 模型不同层特定通信功能外，通信系统操作还需要许多其他管理功能。为了实现这些功能，通信模式通过网络管理扩展其所有层。

例如，网络管理处理操作参数设置和网络节点微控制器操作模式控制。一方面，这包括微控制器不同操作状态转换，以降低电流蠕变影响。另一方面，还包括通过设置网络节点激活或关闭状态进行网络片断操作。此外，网络管理监控参与通信的网络节点。网络管理还为处理过程提供错误记录和报告，并依据不同的应用场合对错误报告给出响应。

举例：遵循 OSEK-NM 节点监控

OSEK-NM 通过逻辑环执行节点监控（图 2-51）。其通过从一个网络节点传递一种特殊类型报文（称为标记）到下一个网络节点（也就是每次都从逻辑前驱到逻辑后继）来完成节点监控。当标记顺着逻辑环传递一周时，就可以确定所有节点是否工作正常。如果标记在规定时段内没有被定义的特定节点接收，则该节点网络管理确定其存在故障。应对该故障节点做出适当响应。

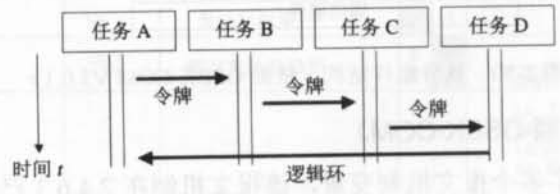


图 2-51 节点监视的逻辑环（依据 OSEK-NM V2.5.2）

2.5.6　总线仲裁机制

多个节点试图在总线上发送报文时，总线仲裁必须要有明确规定。在许多解决总线访问冲突方法中做出正确选择主要是决定实时系统的通信系统调配。因此，本部分介绍一种典型机制。图 2-52 描述了一般应用的总线访问机制及其名称。



图 2-52　总线接入的策略分类

2.5.6.1　总线访问机制——集中或分散执行

总线访问机制一般可以分为两种类型。一种机制为由一个中心网络节点（称为主站）执行。另一种机制为分散执行。在集中执行情况下，使用主/从结构，而在分散执行时使用多主站结构。尽管主/从结构易于实现，但主站一出故障就会瘫痪整个通信系统。通常，多主站结构实现成本高，除非无法预料的外界影响，通信系统在单个网络节点故障或关闭时仍能正常工作。

2.5.6.2　总线访问机制——受控或随机

主/从结构一般应用于总线访问受控机制，而多主站结构在受控和非受控（也就是随机）总线访问机制中应用有所区别。

随机总线访问多主站机制，各网络节点一旦激活就对总线进行发送访问。由于可能存在多个节点同时进行总线访问，该机制也称为载波信号多路访问机制（Carrier Sense Multiple Access，CSMA）。

根据该种机制在总线上是否存在造成冲突的可能性，可以分为冲突机制和非冲突机制。那些尽管不能避免冲突但提供检测和处理冲突方法的机制称为 CSMA/冲突检测（Collision Detection）机制，简称 CSMA/CD 机制。最著名 CSMA/CD 机制应用例子就是以太网（Ethernet）[48]。

能够避免冲突的机制为 CSMA/冲突避免（Collision Avoidance）机制，简称 CSMA/CA 机制。这种无冲突总线访问可以通过一些方法（例如使能网络节点临界同步总线访问）来

实现。这一过程发生在仲裁阶段（Arbitration Phase）前期有效载荷数据传递前。基于传输报文优先级，只有具有最高优先级报文的一个网络节点允许访问总线。本文中，用网络节点优先级代替报文优先级。最著名报文优先级 CSMA/CA 机制应用例子就是 CAN 总线[2]。

在受控总线访问多主站机制中，可分为标控（Token-Controlled）方式和时控方式（Time-Controlled）。

标记是一种有特殊属性的报文，依次从一个网络节点传递至下一个网络节点。当接收到标记时，网络节点被允许在定义时间段内访问总线发送报文。时间段结束后，该节点就将标记传递给逻辑后继。

在时控方式中，每个网络节点都有一个总线访问固定时窗。因此，这种方法称为时分多路访问机制（Time Division Multiple-Access），简称为 TDMA 机制。其实际应用有 FlexRay[49]、时钟触发协议（TTP）[50]和时钟触发 CAN (TTCAN)[51]。

联系现有总线技术和所讨论的机制，我们发现其与实时系统处理器调度十分相似，也可以通过联合机制来实现。相应例子如 FlexRay，一种使用固定时窗随机总线访问 TDMA 方式。

2.5.6.3 总线访问机制——事件驱动和时钟控制

选择一个适当的通信系统需要考虑到多方面的因素，也就是，传输性能要求、安全和可靠性要求及网络空间扩展能力要求。事件驱动访问机制和时钟控制访问机制的实时操作特征存在区别。如 2.4.4.6 部分所介绍过的，这种区别不仅决定网络节点执行任务所需时间间隔的预估性，而且定义给定通信系统传输一个报文所需时间周期（称为通信等待时间）。例如，事件驱动系统如 CAN，能够评估最高优先级报文等待时间。而时钟控制系统能够评估所有报文等待时间。

对于多处理器时钟控制多主站结构，要求具有系统范围全局时钟，或时基。本地实时时钟可能因为微小偏差而不同步（图 2-53），因此所有网络节点实时时钟要有同步机制。且所有网络节点通信系统和实时系统必须支持这些同步机制。OSEK-TIME [17]定义了多种标准同步方式。

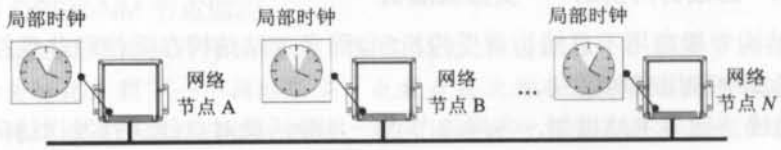


图 2-53 网络中局部系统时钟的偏差

2.6 系统可靠性、安全性、监控和诊断

尽管现代车辆上拥有多个车载控制系统，一个简单功能如制动或操纵系统的失误，就可能造成惨痛的伤亡事故。因此，对于车辆功能的技术实现，在可靠性和安全性方面有非常严格的要求。

因此，提高道路安全性的电子系统开发（如电子稳定程序（ESP），其故障可能会导致

危险情况）要特别考虑应用安全性需求。

电子系统还应用于提高驾驶员辅助程度的车辆功能，在处理过程中承担更多的“职责”。

举例：汽车中与安全性相关的车载电子系统功能在不断增加

- 情况分析，如车速、油量、发动机和车外温度显示；
- 情况评估，如恶劣天气状况警告；
- 动作推荐，如导航系统；
- 动作执行，如干预加速和制动动作的自适应巡航控制（ACC）系统或纠正操纵动作的主动转向系统（AFS）。

鉴于所有这些原因，车辆功能的可靠性、安全性、监控和诊断具有重要意义。当设计与安全性相关的电子系统时，必须对于分布式网络系统的特征进行研究和预测，如实时行为的预测，及子系统和元件错误故障特征分析。同时也要求具有错误快速检测能力，适合系统、子系统及产品和服务环节的诊断程序。因此，本部分介绍可靠性、安全性、监控和诊断技术的基础知识。其他的方面，如法律的限制条件方面的讨论超出了本书的范围。

2.6.1　基本概念

对于车辆功能的需求，主要分为可靠性、可用性和安全性。可靠性和可用性在 DIN 40041 和 DIN 40042 标准中定义，而安全性在 DIN 31000 标准中定义，定义如下[52]：

可靠性　是指一个被分析对象所具有，在一定的时间段内，在给定的条件下，完成指定的功能的特征。

可用性　是指一个系统在指定的时刻，处于可使用状态的概率。

安全性　描述风险不大于限制风险的情形。限制风险一般认为是最大的可能出现的风险。

此外，必须区分错误或缺陷（fault or defect）、失效（failure）、故障（malfunction）：

错误或缺陷（fault or defect）　是指一个观测单元的至少一个性质或特征出现了不允许的偏差。错误或缺陷被认为是一种状态。不允许偏差是指系统的特征或性质的实际值和给定值之间的偏差超过了允许偏差的最大范围。

不同的错误或缺陷机制存在区别，如设计缺陷及结构、组装和制造缺陷。其他类型错误或缺陷包括维护、硬件和软件错误或缺陷及操作者误差或过失操作。错误或缺陷可能但不一定会削弱被观测单元的功能。错误或缺陷可能会导致错误或故障的结果。

失效（failure）　由于本身存在的原因，一个被观测对象在启动后，不能执行指定的任务，因此就不能完成指定的功能。失效被认为是一个事件。失效是一个或多个过失（错误或缺陷，fault or defect）导致的结果。

根据表现不同，失效可以分为多种类型：

- 基于错误频率，如一次、多次和连续错误；
- 基于可预测性，如不依赖于操作时间和其他错误的发生而发生的不可预测错误；
- 系统错误，如依赖特定作用变量而发生的早期失效或磨损故障；

- 确定性错误，如特定情况下可预测的错误；
- 基于错误而引起的损害的大小和程度；
- 基于错误行为的时间特征，如突然故障和老化故障。

故障（malfunction） 在操作启动后，发生的暂时性的错误。一个功能故障是暂时性功能中断或损害。因此就不能完成指定的功能。

2.6.2 系统的可靠性和可用性

可靠性是在特定时间周期内实现预期功能的能力。错误和故障，两者为错误或缺陷所导致的结果，将会损害系统的可靠性。因此，可以通过设法降低错误和故障的发生，来提高系统的可靠性。

基于统计模型的方法是非常合适用来系统地处理与可靠性相关的问题[53, 54, 55]。

平均故障间隔时间（MTTF）、可靠性函数 $R(t)$ 和错误率 $\lambda(t)$，是几个重要的描述可靠性特征方面的统计变量。

2.6.2.1 可靠性函数 $R(t)$ 和错误率 $\lambda(t)$ 定义

研究大量的观测单元数目 $i = 1, 2, 3, \cdots, N$。观测单元 i 的错误特征可以通过时间 T_i 来表示，这期间观测单元处于正常工作（图 2-54）。T_i 称为观测单元 i 的无故障时间（TTF）。

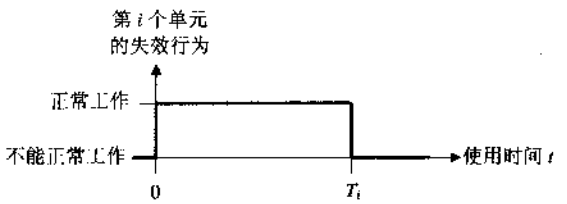


图 2-54 故障发生时间的定义

通过同样条件下的多个相似观测单元，可以获得相关故障频率 $\hat{F}(t)$，使用下面的公式：

$$\hat{F}(t) = \frac{n(t)}{N_0} \tag{2.6}$$

这里，$n(t)$为在时间 t 以后，出现故障的观测单元数目，N_0 为在 $t = 0$ 起始时刻的观测单元数目。$\hat{F}(t)$ 也称为经验故障函数。

这样，经验可靠性函数 $\hat{F}(t)$ 就定义为

$$\hat{R}(t) = \frac{N_0 - n(t)}{N_0} = 1 - \hat{F}(t) \tag{2.7}$$

依据大数定律，$N_0 \to \infty$时故障频率 $\hat{F}(t)$ 就转换为故障概率 $F(t)$。相应地，故障概率补集就为可靠性函数 $R(t)$：

$$R(t) = 1 - F(t) \tag{2.8}$$

因此，$R(t)$表示观测单元在 0 到 t 时间间隔期间处于能够正常工作的概率。常常使用故障率 $\lambda(t)$来代替可靠性函数 $R(t)$。其在可靠性和安全性分析中具有重要功能。

经验故障率 $\hat{\lambda}(t)$ 定义为时间间隔$(t, t+\delta t)$期间故障次数与时刻 t 无出错观测单元个数之比：

$$\hat{\lambda}(t)=\frac{n(t+\delta t)-n(t)}{N_0-n(t)} \tag{2.9}$$

当 $N_0\to\infty$ 且 $\delta t\to 0$ 时，经验故障频率 $\hat{\lambda}(t)$ 趋向于故障频率 $\lambda(t)$，后者可以使用前面所定义的可靠性函数 $R(t)$来表示：

$$\lambda(t)=-\frac{1}{R(t)}\cdot\frac{\mathrm{d}R(t)}{\mathrm{d}t} \tag{2.10}$$

如果故障频率$\lambda(t)=\lambda$=常数，则可靠性函数相应为

$$R(t)=-\frac{1}{\lambda}\cdot\frac{\mathrm{d}R(t)}{\mathrm{d}t} \tag{2.11}$$

或

$$R(t)=\mathrm{e}^{-\lambda t} \tag{2.12}$$

这种情况下，故障概率服从统计学指数分布。

在一些情况下，故障频率$\lambda(t)$随着时间而改变。图 2-55 所示为典型变化过程，根据其形状也可以称为盆形寿命曲线。这种类型故障概率涉及到的概率论分布为韦伯分布。

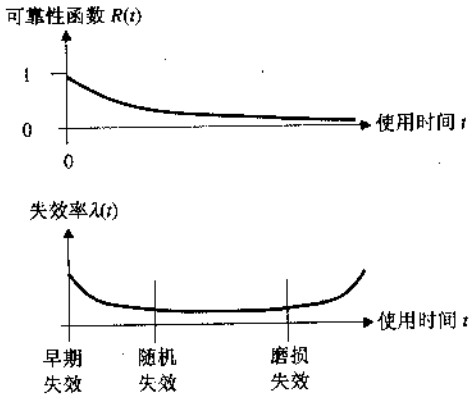


图 2-55　可靠性变量的定义

举例：故障频率经验判定

一千个微控制器在相同条件下同时测试 1000 小时。测试以大约恒定的故障频率产生 10 次故障。那么故障频率为多少呢?

使

$$N_0=1000$$

且

$$n(1000\mathrm{h})=10$$

计算出

$$\hat{R}(1000\mathrm{h})=\frac{N_0-n(1000\mathrm{h})}{N_0}=\frac{990}{1000}=0.99$$

使

$$R(1000\text{h}) = \text{e}^{-\lambda \cdot 1000}$$

得出故障频率

$$\lambda \approx 1 \cdot 10^{-5}\,(\text{故障每小时}) = 10 \times 10^{-6}\,(\text{故障每小时})$$

2.6.2.2 平均故障间隔时间（*MTTF*）

观测单元数目 N 很大时的平均故障间隔时间，计算如下：

$$MTTF = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} T_i \tag{2.13}$$

当故障频率为常数时，计算如下：

$$MTTF = \frac{1}{\lambda} \tag{2.14}$$

举例：*MTTF* 经验判定

30 个微控制器以每小时 10^{-6} 次故障的不变故障频率配置于一台车辆中。假定可以容忍一个微控制器出错，那么 *MTTF* 值是多少呢？

使

$$N_0 = 30$$

且

$$n(MTTF) = 1$$

计算出

$$\hat{R}(MTTF) = \frac{N_0 - n(MTTF)}{N_0} = \frac{29}{30}$$

使

$$R(MTTF) = \text{e}^{-\lambda \cdot MTTF}$$

得出 *MTTF* 值

$$MTTF \approx 3.4 \cdot 10^4\,\text{h} \approx 3.87\ \text{（年）}$$

2.6.2.3 平均维修时间（*MTTR*）

在可维修系统如车辆系统中，必须考虑到故障时间，也就是说，无故障操作时间或正常运行时间 T_B 和不正常运行时间 T_A（图 2-56）。

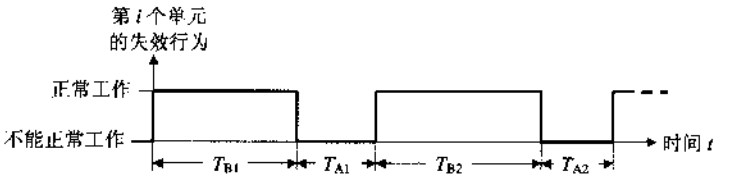


图 2-56 可维修系统的无故障运行时间和故障时间的定义

因此，平均无故障操作时间由 *MTTF* 表示，平均故障时间由 *MTTR* 表示。后者（也就是平均维修时间）由以下公式计算得：

$$MTTR = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} T_{Ai} \tag{2.15}$$

2.6.2.4　平均可用性定义

平均可用性定义为

$$V = \frac{\text{平均维修时间}}{\text{总运行时间}} = \frac{MTTF}{MTTF + MTTR} = \frac{1}{1 + \dfrac{MTTR}{MTTF}} \tag{2.16}$$

因此，为了获得高可用性值，*MTTF* 必须比 *MTTR* 大。可以实现高无故障操作时间 *MTTF* 额定值（例如配置高可靠性元件）和能容纳故障元件的系统结构。例如，配置冗余系统元件提供容错能力。

低故障或维修时间 *MTTR* 额定值可以通过快速且可靠的过失诊断（例如通过检测服务期间诊断支持）或快速且可靠的故障检修（也就是通过简单的维修程序的帮助）来获得。

举例：发动机管理车载诊断（OBD）要求

一些车辆功能可靠性要求通过立法形式强制执行。最著名的例子就是关于发动机废气排放部分的车载诊断系统（OBD）要求。这些要求对发动机 ECU 功能具有非常重要的影响。所有与 ECU 相连的有关排放元件都处于连续监控状态下，且 ECU 自身也处于相同状态下。必须识别、存储和显示错误和故障[56]。

2.6.3　系统的安全性

与定义系统的可靠性和可用性时不同，在定义系统的安全性时，并没有提及被分析对象的功能。换句话说，从系统的安全性角度来看，观测单元是否处于能够工作状态或者处在不能工作状态是不重要的——当然，不能表现出不合理的高风险。

对于汽车车载系统，意味着当只有当无过失状态或者过失状态的后果所引起的危险性处在一个很小的甚至是可以忽略的情况下，才会被认为是安全的。这一无关紧要的可以忽略的风险是可以接受的。提高系统安全性的主要措施是减少由于过失、错误和故障引起的危险结果。

2.6.3.1　安全技术名词定义

常用的关于安全技术方面的名词在 DIN 31000 标准中有定义。

安全工程使用简短而概括的名词损害（Damage）来表示过失、错误和故障造成的消极后果。另一种定义，损害被认为是表示由于特定技术处理或状态造成违反合法保护权利带来的不利条件。确实涉及到的合法保护权利包括，除了人身健康外，还有实物如财产。然而，尽管常常包括生态破坏，但安全工程一般纯粹考虑经济利益，而不考虑生命和人身损害。

风险　安全风险，简称为风险（也就是危险量化），并不能被完全排除。安全工程一般定义风险为造成损害的事件发生概率和事件造成的损害程度的乘积。这两个特征量描绘了测量风险的方法。常用的描绘风险的方式为多维变量。导致损害的事件称为事故

（Accident）。因此，使用下列式子：

$$\text{风险}=\text{事故发生概率}\times\text{事故造成的损害程度} \tag{2.17}$$

或者是，

$$\text{风险}=\begin{pmatrix}\text{事故的概率}\\ \text{事故造成的危害程度}\end{pmatrix} \tag{2.18}$$

风险的限制值 表示最大正当风险。一般认为限制风险不能量化定义。因此，由特定安全要求间接表示。这些要求包括适用于个案的全部定律、指示、指导方针、标准和规则，且隐定义了限制风险。

危险 表示一种对人类和（或）环境存在实际或潜在威胁的情形。这种危险可能导致对人类、环境及观测单元自身具有消极后果的事故。因此，危险建立了一种风险实际超过了限制风险的情形。

各种危险（例如电危险、热危险、化学危险和机械危险）可能由多个系统引发，也就是说，由一个作为整体的给定系统，而不是单个部分引发。因此，在多数情况下，实际上不可能事先识别和避免（不考虑预防）所有不同类型危险。因此，一般认为存在（任何系统）一点残余危险是可以接受的。识别这些危险是危险分析学科的目的。

安全 表示风险小于限制风险的情形。

图 2-57 以绘图形式形象化地表示了四个基本名词之间相互关系：风险、限制风险、危险和安全。

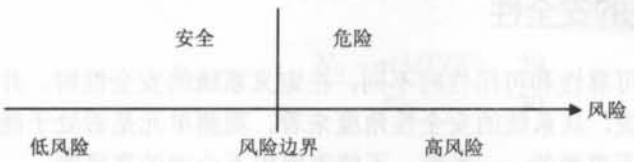


图 2-57 安全工程中几个基本术语间关系的图示

名词保护（Protection）的定义和风险的定义十分相似。保护通过限制发生概率（Incidence Rate）或损害程度等措施来降低风险。

2.6.3.2 风险的确定

风险分析观测特定系统故障和错误风险，其按照 DIN 19250[18]和 IEC 61508[19]标准执行。该规范使用以下参数变量：发生率 W，损害程度 S，不工作状态 A 和危险预防 G。

如图 2-58 所示的风险分析图，依据 DIN，利用这些参数用于确定不同的要求种类 AK0 ~AK8，或依据 IEC 利用这些参数用于确定不同的安全完整性水平（Safety Integrity Level，SIL），SIL0~ SIL4。这两个参数为风险测量特征量。

风险分析必须观测所有的系统功能及评估其单独或整体潜在危险。这包括通过使用适当的风险参数对系统提供的每个功能进行可能故障评估。评估结果作为设计系统适当结构的基础。

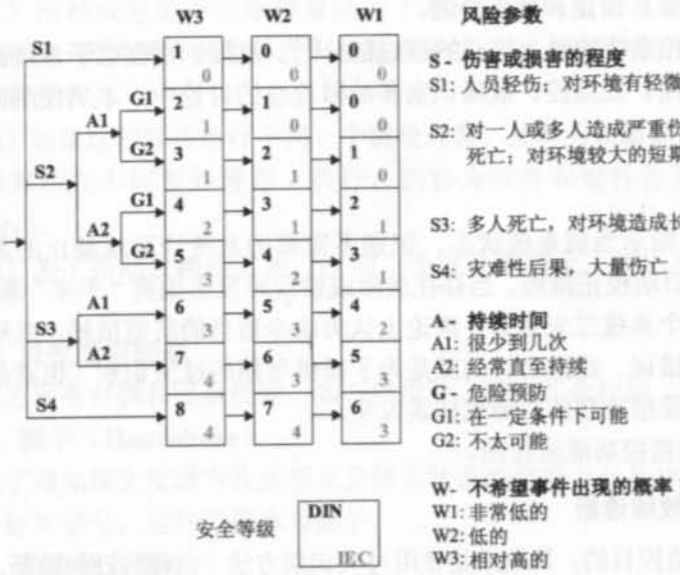


图 2-58　风险分析图和安全要求的分类

举例：确定电子节气门控制（ETC）系统的要求种类

定义 ETC 系统（Electronic Throttle Control System，即电子节气门控制系统）的要求种类。图 2-59 所示为汽油发动机 ETC 系统构成的简化图。

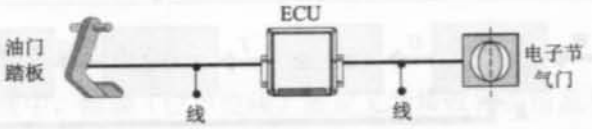


图 2-59　汽油发动机的 ETC 系统

本例中，假定为临界驾驶情形。

- 驾驶情形：高速行驶中的车队
- 可能危险：完全无意识加速，导致追尾或车辆失去控制
- 风险参数：S3——多人伤亡
 A1——几乎不发生到偶尔发生
 W1——非常低的发生率

这一评估对加速功能得出的要求种类分别为 AK4 和 SIL2。这种分类成为诸如 IEC 61508 标准定义安全要求的基础，调整系统结构（诸如硬件、软件、设定点发生器、传感器和执行器）。

2.6.4　系统监控和诊断

如果一个与安全性相关的系统，不能可靠地完成它的功能，并因此而可能出现危险。在这样的情况下，就要求按照一种定义好的安全逻辑程序做出处理。解决安全性问题的前

提是，必须进行可靠的故障、错误和过失检测。

因此，故障识别是监控系统的最为核心的组成部分[52]。故障识别在电子系统的可靠性和安全性方面起重要的作用。在监控、故障识别和故障检修的讨论中，本书使用参考文献[52]、[57]中的名词和定义方法。

2.6.4.1 监控

技术系统监控是为了指示当前系统状态，识别不需要的系统情形或禁止的系统情形（如系统过失），且需要时启动校正措施。当存在故障或错误时发生偏离“平常”系统状态，这种现象可能由一个或多个系统过失造成。理论上认为缺少适当的防范措施，系统过失会导致短期或长期的故障或错误。监控的目的就是为了提供早期的过失识别（也就是在故障或错误发生前）及采取必要措施预防故障或错误发生。

图 2-60 所示为典型的监控功能流程图。

2.6.4.2 故障识别和故障诊断

前面部分所讨论过的监控目的，其结果是使用过失识别方法（也称为过失诊断，或简称诊断）来检验两个确定数值最小值之间是否存在相关性。超出规范的偏离归成几种过失症状。

电子系统过失识别措施或过失诊断措施的例子如下。

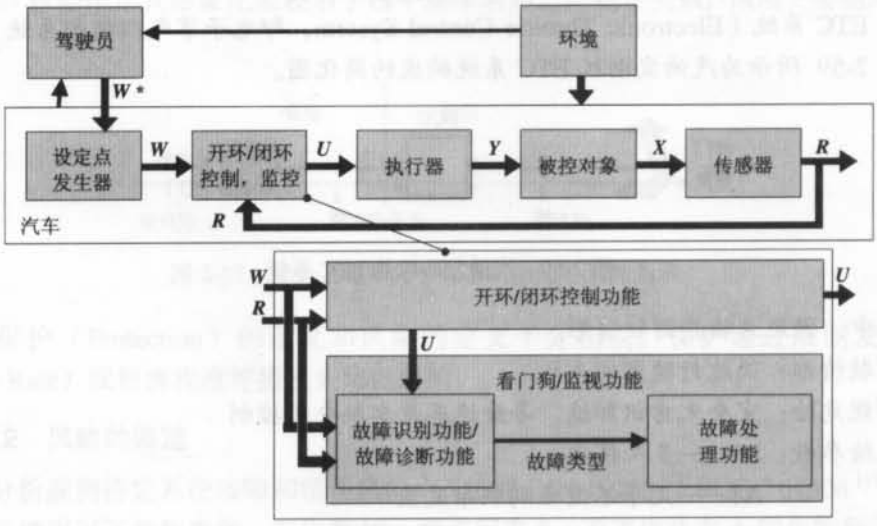


图 2-60 监视功能图示

1. 参考值校验

一个问题（获得已知响应）摆在我们面前（询问/响应规则）。为了确定响应，系统必须执行相同功能或用于正常操作的子功能。如果获得的响应未能达到已知响应，则这就解释为过失。

2. 冗余值校验

可以用到两个或更多的比较值，且其比较结果有助于发现误差。有多种软件实现方法来实现这一功能：

（1）两种或更多不同原理算法用于相同的输入值。基于不同原理的性质就是差异（Diversity）。因为所有的软件过失都是系统级过失，所以基于软件的过失识别对这种差异具有强制要求。

（2）如果这些算法运行于同一个微处理器，就称为软件差异（Software Diversity）；如果这些算法在不同微处理器上执行，则称为软件和硬件差异（Software and Hardware Diversity）。

（3）为了帮助发现伪过失，在同一个微处理器上，使用不同输入值重复运行同一个算法。

3. 监控通信链路

这方面有奇偶和冗余校验，如奇偶校验、循环冗余校验（CRC 校验）及海明码[58]。

4. 握手（Handshake）

为了通知报文发送方收到报文及报文状态或接收方自身状态，报文接收方向发送方发送一个证实信号。这种通信称为握手。

5. 监控物理属性

典型例子是温度传感器，其高测量读数指示过失传感器信号。另一种应用是根据是否保持在一定的限制值内来校验信号值，以及校验信号值随时间的变化情况。

6. 监控程序执行

例如，可以通过看门狗电路实现对一个过长的程序执行时间错误的响应，如微处理器复位。

2.6.4.3　错误检测和纠正

在数字通信系统中，错误（位单位级）被定义为接收到的信息与发送方发送的信息不匹配的情况。本文中，有助于了解过失（其可能在系统中出现）及其可能的根本原因（可能是无效传输成分或软件过失，称为错误）之间的区别。因此，用于查找基于软件起因问题的程序方法称为错误检测方法。

错误检测和纠正包括以下几种措施：

（1）冗余值校验。错误纠正，冗余（或奇偶）校验方法与错误检测方法相差无几。为了确保错误纠正，必须定义一种标准从而能够确定和应用校正值。有多种实现方法：

- 错误检测已经提供了确认错误结果的信息；
- 在一些情况下，可以通过应用容错算法（也就是“安全可靠”运行）来响应过失。例如使用第一个或较高的值、平均值或类似的算法。

（2）关闭子系统或切断主系统。

（3）保持过失状态或启动机制的变化。

（4）过失存储，如 ECU 单元过失存储器（详见 2.6.6 节）。

（5）过失校正，如通过看门狗模块复位微处理器。

也可以联合应用以上措施。

2.6.4.4 安全逻辑

安全逻辑定义应用于安全相关系统的错误纠正措施。分为几种系统类型。

作为第一步，定义安全状态（Safe Sate）。

在能够采取这种安全状态的系统中，如定义的紧急关闭（Emergency Shutdown），也常被称为紧急停车（Emergency Stop），其安全响应就是启动这一安全状态。必须确保这一安全状态只有在一种控制方式下才能退出，即并非其他过失、故障或错误所导致。具有这种安全响应特征的系统也被称为故障保护系统（FS 系统）。

在一些情况下，采用安全状态后系统进入低性能操作模式（Degraded Operating Mode）。这时继续（虽然有限）系统服务，如软返回（Limp-Home）操作模式，被称为故障减少系统（FR 系统）。当然，存在由于资源不足或风险最小化而有意引入低性能操作模式的情况。

有时技术障碍妨碍了安全状态的引入。例如，汽车行驶过程中的一些车辆托管功能应用。如果系统突然出现故障，则必须压制系统错误对车辆动作的不利影响，且需要切换到适当的备份系统。原则上，所执行的备份系统可能与出现故障的系统相同或有所不同。具有这种安全响应特征的系统称为故障操作系统（FO 系统）。

例如，基于安全逻辑的系统要求一般为 FO/FO/FS 或 FO/FO/FR 情形。这意味着这些特征系统必须保持完整操作，即使出现 2 次连续内部故障。而出现第 3 次故障时，系统则进入安全状态和（或）进入“Limp-Home”操作模式。

当制造商申请车辆登记首先需要的一般类型批准时，第一步就是要进行“简单”的安全验证。这就要求使用简单基本且易于掌握的概念。因此，简单是相关安全系统的基本设计原则之一[57]。

另一个设计目标就是尽可能使全部元件符合严格的安全要求。因此，封装和部件结构要符合相关安全系统基本设计原理。

对于其他开发内置故障和错误容许系统的原理和方法的讨论已经超出了本书的范围。这些原理和方法中，在这里主要提到，故障树分析（Fault Tree Analysis）、因果分析（Cause-and-Effect Analysis）和错误模式和结果分析（FMEA）。关于这方面的更多信息，请参考文献[59]、[60]、[61]。

2.6.4.5 功能软件的安全性

关于功能安全性，我们要特别注意几门经典学科开发和设计分支之间的巨大差异，如机械工程、液压工程和电子工程，及新产生的软件工程[59]：

（1）在开环和闭环控制系统中，软件经常承担原先用模拟技术来实现的开环或闭环控制器功能。控制器功能从模拟转换为数字是比较困难的，且可能产生误差。一般情况下，连续函数转换为离散函数是比较费力的。这是因为离散函数的规格要比其模拟信号复杂得多。如值离散化、时间离散化和拟并发处理功能。

（2）与软件测试不同，模拟系统的物理连续性有助于简化模拟系统的测试。一般情况下，物理系统工作在某个固定的区域范围内，并且在导致最终的故障之前，都会出现物理失真的现象。在多数情况下，一个小小的情况变化就可能导致行为上的一个小的变化。这

些情况使得可以在工作的范围内在几个特定的工作点进行少量的测试，并且依据系统的连续性，和利用内插法和外推法等方法来估算其他工作点的情况[30]。这种方法完全不同于软件，软件方法可能因为输入变量情形的一点差错而产生故障。此外，故障软件可能出现与标准执行完全不同且不可预知的后果。

（3）对物理系统大的改动通常只有花很大的财力物力才能实现。这些也并不适用于软件，因为其不受这些“物理上”的约束。

（4）软件并不会因为老化而出现故障。其质量也不会受到产品和服务方面的影响。无一例外，软件缺陷在开发阶段已经存在。因此，要非常注意软件设计过程，才能保证质量。

在可靠性软件的开发过程中，必须考虑以下几个重要的方面：

（1）软件开发必须确保系统能够在指定的各种可能的情况下都能正确地执行。

（2）必须及时检测和纠正运行时的错误，如故障错误造成的偏离；同样对于在软件开发阶段没有发现的错误必须能够及时地识别和处理。

（3）必须采取适当的安全措施来发现和阻止对系统的操作，如未授权的 ECU 程序和数据。

2.6.5　电控单元监控系统的构成

ECU 监控系统一般通过结合硬件组成和软件措施来实现。硬件组成方面如配置智能输出模块或看门狗电路。监控功能的软件实现使得可以应用一些灵活的概念和方法，以对过失、故障和错误做出响应。

与纯粹的机械或液压实现组成相比，结合电子组成后的系统的优点在于能够提供过失、故障和错误（这些并不只出现在机械和液压组成中，也出现于电子组成中）的检测和检修。

鉴于以上这些原因，一些当代应用系统把监控系统功能的设计作为与控制系统功能设计处在同等重要的位置来看待。可以这么说，持续增加的监控功能软件实现对整个电子系统和软件结构设计有着重要的影响。因此，在开发早期及整个开发阶段考虑监控功能的开发是必要的。鉴于监控功能的重要性，本文后面提及的功能需要结合开环/闭环控制功能和监控功能来理解。

本部分讨论软件监控系统，一般配置于 ECU 中。如图 2-61 所示，监控系统分为两层。底层处理微处理器的监控任务。上层实现设定点发生器、传感器和执行器的监控功能。

在多数情况下，微处理器监控使得有必要使用第二个计算机，称为监控计算机。作为这种情形，用于微控制器监控的软件功能可以由功能计算机和监控计算机来实现。因而两个计算机彼此制约。

如果发现故障，两个软件层的功能计算机和监控计算基本就会触发适当的矫正动作。并且这些矫正动作可能以硬件或软件功能形式执行。最后，过失症状可以作为监控功能的初始值。

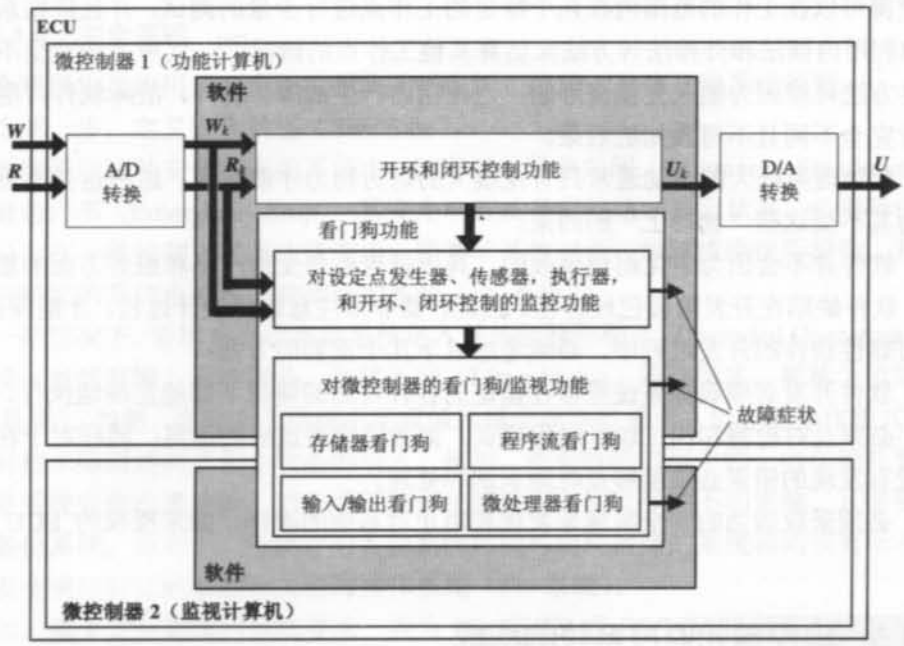


图 2-61 ECU 中的软件监视系统

2.6.5.1 微控制器的监控功能

微控制器监控功能检测各个组成部件，如微控制器自带的存储区域（例如，Flash、EEPROM 或 RAM）、输入/输出单元或微处理器。其中一些检测是 ECU 供上电后的启动程序。为了确保甚至是在运行操作中发现元件故障，一些检测程序在 ECU 执行正常功能时重复运行。有些检测程序要求具有相对较灵活应变的计算时间（如 EEPROM 检测），在关闭后执行，也就是说在车辆停放时执行。这样能够避免车辆在运行中处理功能冲突所需的时间延迟。

除此之外，还监控程序执行。例如，该功能检验任务是否按预定激活执行，或检验所需报文（也就是在 CAN 总线上传输）是否在预先规定时间内到达。更多关于实时系统误差检测与误差纠正的详细讨论，参看本书第 5 章 5.2.2 节。

2.6.5.2 设定点发生器、传感器、执行器及控制功能监控

例如，设定点发生器和传感器的监控功能检验连续性和真实性。这可以通过不同设定点发生器和传感器信号之间的物理关联来实现。不真实的信号值将导致错误处理程序，如信号值迫使进入 Limp-Home 操作模式。

同样，也要监控执行器功能和连线。一方面，这要求发送测试信号并检测响应结果。显然为了防止危险情况发生，该程序过程必须在特定情况下执行。另一方面，供电期间执行器电流值与预存电流限定值相比，相应过失偏离可以通过指示器显示。

另一个监控的应用领域就是监控控制功能的计算。例如，控制功能的计算输出值通常依据一个简化的监控功能的输出值来测试控制功能计算输出值的真实性。

举例：发动机 ECU 原理图

严格的安全性和可靠性要求迫使一些发动机管理功能在发动机 ECU 中配置一个监控计算机（图 2-62）。发动机 ECU 相关安全功能监控概念详细说明及分析方法，参看第 5 章 5.2.4 节。

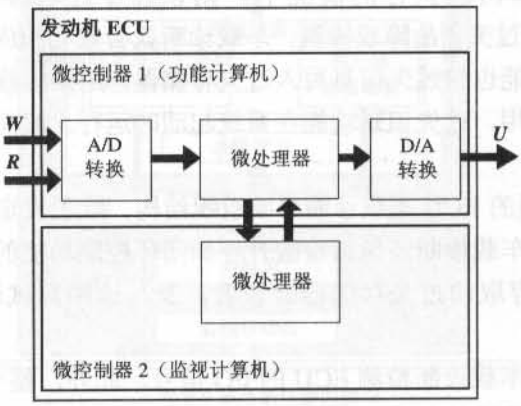


图 2-62　发动机 ECU 的简化方块图

2.6.6　电控单元诊断系统的构成

诊断系统（作为监控系统的子系统）是产品 ECU 基本结构的一部分。可以分为车载诊断和非车载诊断（图 2-63）。

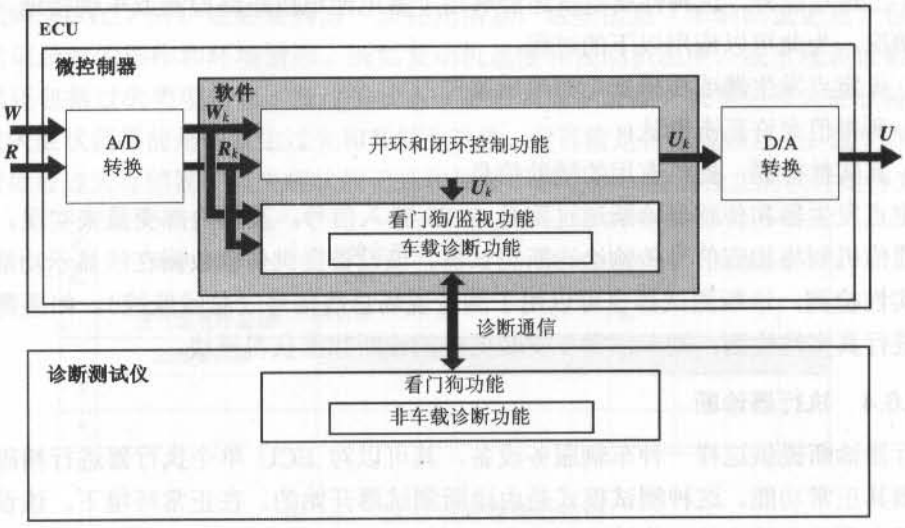


图 2-63　车载和非车载诊断功能

2.6.6.1　非车载诊断功能

只要故障诊断是通过将 ECU 和诊断测试器互连而执行的，则该程序过程称为非车载诊断（Offboard Diagnostics）。通常，将维修车间的诊断测试器与车载中心诊断插口连接

器相连。通过该连接器可以与车辆整个 ECU 网络互连。这样，所有可诊断 ECU 都能得到诊断。

2.6.6.2 车载诊断功能

故障诊断在 ECU 内部执行的情况下，则该程序过程称为车载诊断（Onboard Diagnostics）。如果发现过失、故障或错误，车载诊断就会起动相应的故障检修程序。在多数情况下，车载诊断功能也将过失信息写入过失存储器，用于维修车间诊断测试器的读取和评估。根据各自的应用，过失识别功能在系统起动时运行。然而，也可能在标准系统操作期间循环执行。

图 2-64 所示为典型的 ECU 车载诊断系统组织结构。除了设定点发生器、传感器和执行器诊断软件功能外，车载诊断还包括检验开环和闭环控制功能的程序。诊断系统还包括处理过失存储器读/写存取的过失存储器管理者，及与诊断测试器通信的车外诊断软件平台。

在标准操作期间，车载诊断检测 ECU 的 I/O 信号。此外，整个系统一直处于监控中，以防异常行为、故障和错误的发生。

汽车诊断系统最初只是维修车间的简单直接故障检修程序。然而时至今日，随着安全性和可靠性有关法规的推动，诊断系统已经完全成为 ECU 的一个子系统。

2.6.6.3 设定点发生器和传感器诊断

设定点发生器、传感器及其与 ECU 之间的配线互连可以通过评估输入信号来进行监控。除了上述功能外，这种性质测试还能够用于蓄电池电压短路检测或车辆接地，及配线断路等情况。为此可以应用以下的过程：

（1）设定点发生器或传感器电源电压监控；

（2）所得值允许范围确认；

（3）真实性检测，提供有用的辅助信息。

设定点发生器和传感器诊断通过测量 ECU 输入信号，及其内部变量来实现。接着车外诊断通信机制将相应信号传输给诊断测试器。该设备提供传输数据在线显示功能。为了进行真实性检测，诊断测试器也可以用于测量现场总线信号（总线监控）。如果需要对其他信号进行真实性检测，则车辆需要安装更多的诊断数据获得模块。

2.6.6.4 执行器诊断

执行器诊断提供这样一种车辆服务设备，其可以对 ECU 单个执行器进行精确定点激活来检测其正常功能。这种测试模式是由诊断测试器开始的。在正常环境下，该设备只在静止车辆上而且在特定的工作情况下工作。例如在发动机 ECU 中，诊断测试器只在低于特定允许发动机速度或在发动机止转（接通/发动机关闭情形）时工作，这时开环或闭环控制功能并不工作。然而，由 ECU 输出给现场执行器的信号，可以由诊断测试器来模拟。执行器功能可以通过听觉（例如通过电磁阀打击声或执行器动作的拍打声）或其他一些简单方法来确认。

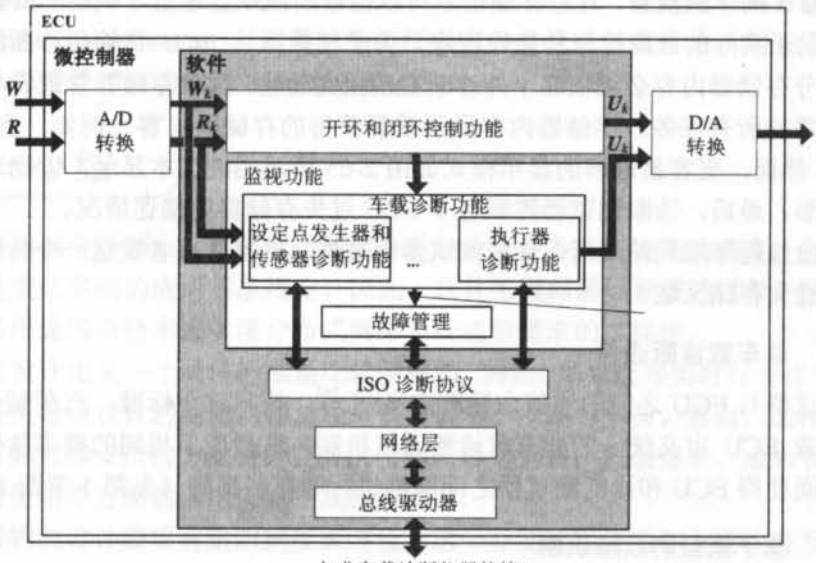


图 2-64　ECU 车载诊断系统示图

2.6.6.5　故障的存储器管理

被车载诊断功能发现的故障症状一般保存在 ECU 的存储器中。在多数情况下，保存故障信息的存储器位于 EEPROM，因为这有助于记入项目的永久存储。图 2-65 所示为一个典型的发动机 ECU 过失存储器结构图表。规定要求详细说明了每一个过失项目除了附上故障代码（DTC）外，还必须列出一些补充信息。这些信息（车辆测试记录）包括指示过失症状记录时的操作和环境情形。例如发动机速度和发动机温度，或里程表读数。其他存储数据还包括过失类型信息。及过失状态（静态或偶发故障）或其他过失症状特征。包括指示过失症状获得前是否发生过失和其发生次数，及目前是否出现该过失。过失存储器的管理负责处理过失存储器中过失症状的记录和恢复，该功能一般由独立的软件组件来实现。

Pos. No.	故障症状	故障代码（DTC）	故障指示灯（MIL）	存储	活动	环境条件		
1	进气温度传感器	P0110	off	yes	no			
2	加速度信息	P1605	off	no	yes			
3								
4								
⋮								

图 2-65　ECU 故障存储器结构

发动机管理一般要符合国家明确规定的废气排放标准。排放标准包括一些过失影响废气排放规范，如诊断故障代码（DTC）和故障指示灯（MIL）闪烁代码，作为驾驶员信息。给定某一指定情形，如驾驶期间未能记录下 DTC 的情形下，过失存储器管理可能会从存储器中再次删除一些过失症状（或故障指示）。

对于维修车间车辆检查，这些存储信息可以由诊断测试器通过车辆的车外诊断接口来获得。这有助于实际的故障检修和修理程序。为了使数据从 ECU 传输至诊断测试器，图 2-65 灰色部分存储器内容必须依靠车外诊断通信报文传输。过失存储器管理者处理诊断测试器使用所需的所有任务。存储器内容显示重新获得的存储器内容（例如，直接以 DTC 形式显示）。然而，更容易理解的显示模式如图 2-65 所示的纯文本显示，以物理变量形式显示环境情形。最后，诊断测试器需要各个 ECU 过失存储器的描述情况。

当故障检修程序顺利结束后，诊断测试器向过失存储器管理者发送一个特殊命令，从而清除整个过失存储区域。

2.6.6.6 非车载诊断通信

诊断测试器与 ECU 之间的通信由标准定义[25, 26]。按照这个标准，汽车制造商一般会为所有的车载 ECU 定义统一的非车载诊断通信机制。这形成了相同的整车软件组成标准化基础，从而使得 ECU 和诊断测试器之间能够进行非车载通信（见第 1 章图 1-22）。

2.6.6.7 基于模型的故障识别

本部分将重点讨论基于模型的故障识别（也称为基于模型的诊断）。这种方法逐渐应用于 ECU 中。图 2-66 为功能结构图。

过失和误差识别利用静态和动态系统行为的各种可测量信号之间的相互依赖关系（通过应用执行器模型、被控对象模型和传感器模型等）。为此可以应用控制工程中的方法，如模型方程、状态变量估算或状态观测器等[34, 35]。基于现有的输入变量，如在多数情况下，控制器输出变量 $\boldsymbol{U}$ 和反馈变量 $\boldsymbol{R}$，在有些情况下，也为参考变量 $\boldsymbol{W}$ 和控制变量 $\boldsymbol{Y}$，可以通过这些输入变量来识别控制器故障、传感器故障、被控对象故障和执行器的故障。

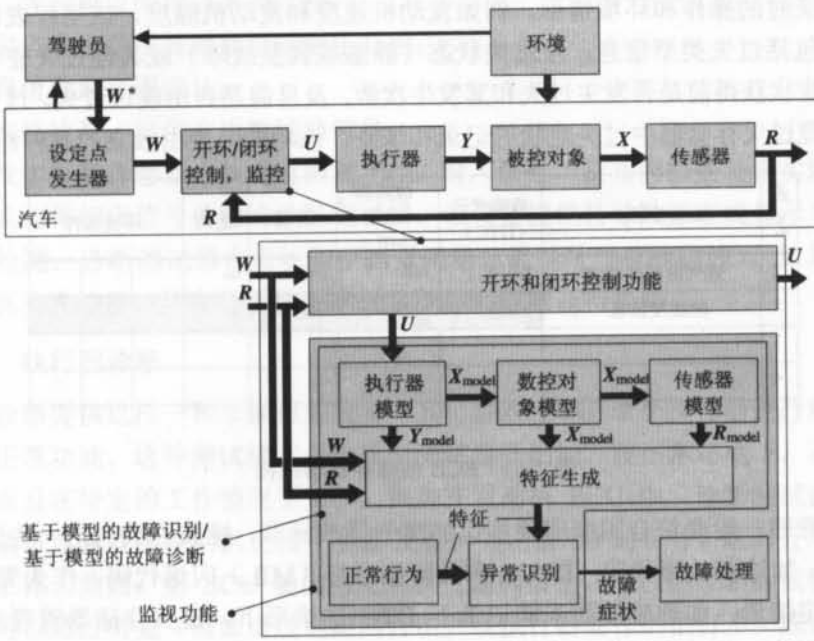


图 2-66 基于模型的故障识别方块图

基于模型的故障识别通过将模型组成与现实组成相比较，使用适当的方法来产生相应特征（Characteristics/Feature）。如果这些特征偏离参考值或正常行为，则鉴别为不同故障的类型，从而进行相应的故障处理和故障检修。

2.7　小结

正如前面部分所预示，汽车电子和软件系统设计任务具有较高的难度，主要是因为各种全面而且变化多端的应用需求规定。因此，软件工程师需要掌握通过单一的、合理的、标准的和多用途网络技术来实现分布式网络系统通信需求的多样性。

为系统设计定义一个最佳标准是比较困难的。例如，如果选择实时行为作为最佳标准，这将可能甚至导致设计的处理器和总线只能提供最小的运行负荷。然而，这种解决方法将不能确保可靠性和安全性方面得到充分的考虑。换句话说，冗余需求、成本和质量保证，如元件重新使用等方面就会出现偷工减料的情况。

复杂事物的设计需要有组织的步骤过程。这一开发过程包括整个设计开发阶段，从需求分析到最终系统验收测试。

一种方法是将具有相同需求规格的子功能合成一组，这在实际应用中已证明是有好处的。各子系统电子系统通过适当的通信技术相互连接，超出子系统范围的专门功能的执行由子系统通过具有网关的控制单元相互连接来完成。

举例：BMW 7 系 ECU 网络[62]

图 2-67 所示为 BMW 7 系的最新 ECU 网络模型图。其大概包括 60 个 ECU，这些 ECU 构成 5 个子系统。

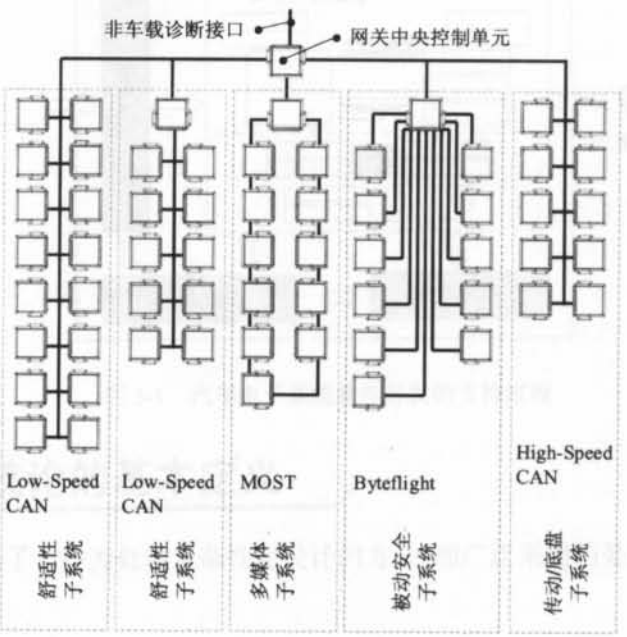


图 2-67　宝马 7 系汽车的 ECU 网络

诊断报告要求通过点对点连接关系从诊断测试器经由中心非车载诊断接口传输给中枢网关控制单元。从该点向前，通信经由内部总线和子系统各控制单元的网关控制单元继续进行。

子系统利用各种通信技术和网络拓扑：

（1）高速变速 CAN 总线[2]配置于车辆动力传动系和底盘区域；

（2）低速变速 CAN 总线[63]配置于舒适和便利系统的总线技术，由两部分组成；

（3）Byteflight 总线[64]星型拓扑结构将被动安全系统控制单元相互连接；

（4）多媒体系统要求高数据速率和精确性，及时间同步数据传输，这些系统通过 MOST[65]环型拓扑相互连接。

在撰写本书时（2005 年），已经出现了或即将出现其他一些通信技术：

（1）在多媒体应用领域中，技术革新周期差异（多媒体设备需要几个月，车辆系统需要几年）导致了无线通信技术的应用，如蓝牙技术[67]，将车载硬件的限制降到了最低水平。

（2）在占成本大部分的车身应用中，引入一种具有成本效益的 LIN[68]分网络技术。

（3）配置相关安全系统，如电子控制的制动（Brake-by-Wire）或电子控制的转向驾驶（Steer-by-Wire）系统，具有容错设计要求的通信系统。相应的网络技术有 FlexRay[49]、TTP[50]和 TTCAN[51]。

第 3 章

Chapter 3

电子系统和软件开发的支持过程

本章将集中讨论汽车电子系统和软件开发的一些支持过程（如图 3-1）。本章首先定义几个有关系统技术的通俗术语。接下来的内容是讨论用于配置管理、项目管理、供应商管理、需求管理以及质量保证的一些过程。因为所讨论的这些内容很大程度上独立于软件开发，所以更易于把这些过程和方法应用到汽车中所有的系统层，以及给设定点发生器、传感器、执行器和其他硬件的开发，还包括软件自身的开发。

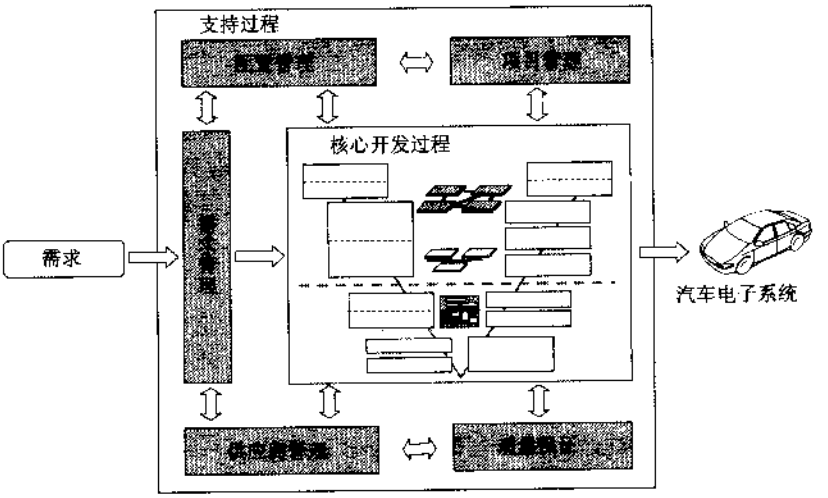


图 3-1　汽车电子系统软件开发的支持过程

3.1　系统论的基本定义

系统论[59]给出了一些为处理复杂性而设计的方法。而广泛采用的处理复杂性的方法可以

看作“分解与控制”。它是基于下列三个被认为是可以明确阐述大量实际问题的重要假设的。

（1）把系统分解成一些组成部分会使得问题容易观察。

（2）独立地看，这些组成部分本质上等同于系统的组成要素。

（3）把这些组成部分组成一个系统的原则都是简单的、稳定的和众所周知的。

形成系统的各组成部分之间的相互关系体现了系统的一些特性。随着系统复杂性的增大，系统的组成部分分析和组成部分间的相互依赖会变得越来越复杂，所花费的代价也会越来越昂贵。而这类系统恰恰是系统论所要致力研究的。这里值得注意的是，一个系统的组成部分可能是完全不同的。系统的组成部分可能是某种技术组件，也可能是人或者环境。

下面几部分主要讨论技术系统。本书是基于下面的一些系统定义[59, 60]而展开讨论的。

（1）**系统** 它由一组相互作用且独立于周围环境的部件组成（图 3-2）。

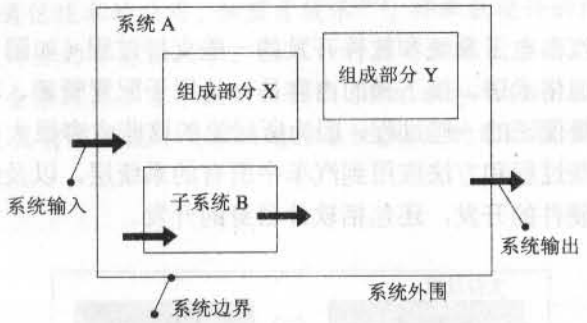


图 3-2 系统的方框图

（2）**系统状态** 在某特定时间点，系统的状态取决于描述系统每一时间点特性的集合。

（3）**系统外围** 也可简写为外围。它描述了一组部件及其特性，这些部件不作为系统的一部分，但其行为可能影响系统状态。

（4）**系统边界** 它描绘了系统和系统外围之间的边界。

（5）**系统接口** 任何越过系统边界的信号就变成了暗含这个动作的系统接口。

（6）**系统输入和系统输出** 它们是系统接口上的特定名称，反映了所处理的数据在向内传输和向外传输上的不同。

（7）**子系统** 因为系统经常是其环境的一部分，所以它可能被认为是组成一个大系统的一个组成部分。因此可以这样假设：被认为是一个系统的任何一组组成部分通常也是系统层的一个部分。这样，一个系统也可能包含几个子系统，也就表示一个系统是其子系统的一个集成。

（8）**系统层** 通常系统论使用几个所谓的观察层，也可以称为系统层。

（9）**自相似** 如果不同的系统层具有相似之处，例如显示了共同的特性，那么就使用这个术语。如图 3-2 描述了系统 A 和子系统 B 之间的相似之处。

（10）**内部观和外部观** 它们描述了从系统内或系统外观察一个系统的不同。总而言之，在外面看里面，总是不可能讲所观察到的是否就表示为一个组成部分或一个子系统。外部观是一个抽象的关于系统边界和系统接口的系统观。

因而任何一个系统观都包含由各自的观察者所分析得到的一个抽象层。这使得不同的

观察者对同一系统可能形成不同的系统观。例如，第 2 章引入的建模观，例如基于方框图的开环控制/闭环控制建模方法，微控制器建模观和安全技术观。所有这些观点都表示了汽车电子系统的不同观察方法。

通常采用的系统建模方法都使用了一些抽象工具，例如层的构成或层的构建和模块化。通常直觉上就把这些基本的原则应用到致力于获得系统观的大批任务中。

对建模有实质帮助的是所谓的 7+2 或 7–2 规则。在许多情况下，包含超过 7 + 2 = 9 个组成部分的系统对人作为观察者来说已相当复杂，但是反之包含少于 7 – 2 = 5 个组成部分的系统通常被认为是容易的。如图 3-3 所示，包含 5 到 9 个组成组成部分的系统看起来是可处理的。

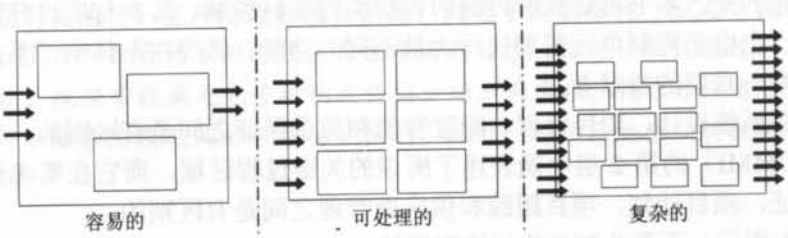


图 3-3　结构的清晰性——系统建模的基本规则

存在于系统和其组成部分之间的确定的包含关系称之为关系集合或集合。把一个系统分解为一些组成部分，就称之为分割或分解。因此，对组成部分进行集成从而形成系统，就被认为是集成或合成。

举例：汽车电子系统的系统层

汽车电子系统可在几个系统层进行观察。图 3-4 标注了几个不同的层，以便和下面的几部分保持一致。

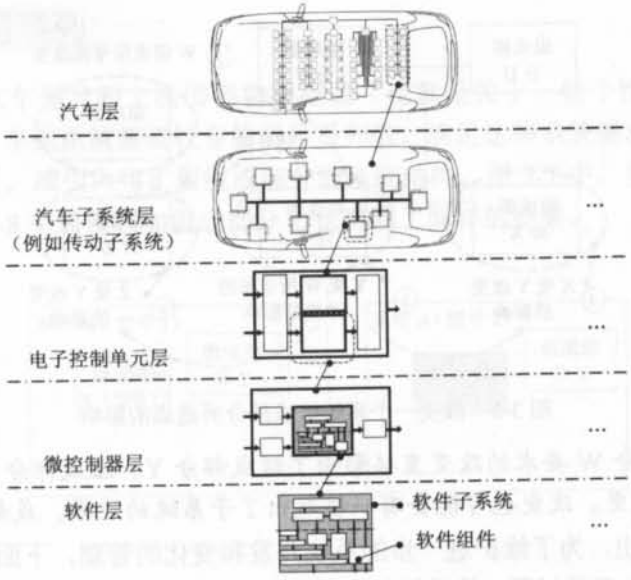


图 3-4　汽车电子系统的层次结构

3.2 过程模型和标准

用于系统开发的需要，人们已经提出了多种不同的开发过程模型和标准，例如能力成熟度模型集成(CMMI)[13]，软件过程改进和性能确定(SPICE)[14]或生命周期过程的V模型[15]。

在应用于某个具体的开发项目之前，必须要有一些有效的方法来评价一个过程模型，以用来确定哪个模型对预期的保证是最合适的。当然原因是多方面的。许多情况下，单个过程步骤的重要程度随着它们的应用而变化。例如，在电子控制元件中，功能校准起着非常重要的作用。但在其他应用领域（例如车体电子系统），功能校准的重要性是比较低的。

应当注意的是，本书第2章所讨论的各个工程学科领域，其参与项目开发的程度是不同的。例如一个电子控制单元需要执行大量不同的功能。然而在车体电子系统中，分布式网络系统是唯一重要的贡献者。

V模型在系统设计、项目管理、配置管理和质量保证之间是有区别的。在能力成熟度模型集成（CMMI）的第2层明确表述了所谓的关键过程区域，而它在需求管理、配置管理、质量保证，项目计划、项目跟踪和供应商管理之间是有区别的。

如图3-1所示，下面几部分将讨论配置管理、项目管理、供应商管理、需求管理，以及质量保证的不同方面。

对于前面所列主题，本章不作全面讨论，而是通过现实案例来证明这些过程的优点。重点指出那些在汽车开发中有重要地位的过程步骤。

举例：进一步开发和改变的管理

由于汽车的产品生命周期很长，因此车载系统的进一步开发和改变的管理等方面都是非常重要的。必须能管理改变对一个系统所造成的影响，以及跟踪这些改变。图3-5显示了一个例子，即改变一个组成部分如何影响系统的其他不同组成部分。

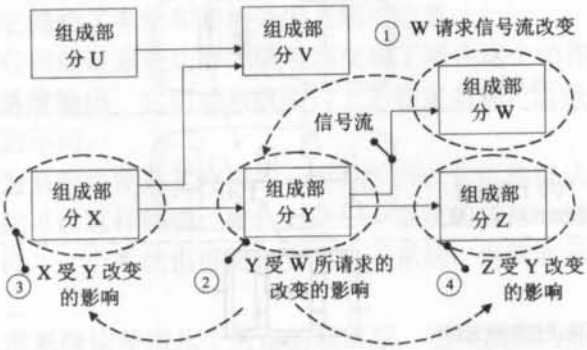


图3-5 改变一个系统组成部分所造成的影响

最初，组成部分W要求的改变直接影响了组成部分Y。组成部分Y的改变导致了组成部分X和Z的改变。改变也可能会有一些超出了子系统的边界，或越过了系统层。

所以有理由指出，为了维护进一步的系统开发和变化的管理，下面几部分要讨论的支撑过程必须连接核心开发过程。这是管理和跟踪系统各组成部分之间横向依赖的唯一方法。

3.3　配置管理

3.3.1　产品生命周期

产品的生命周期包括三个不同的阶段：开发阶段、生产阶段、运行和服务阶段（见第 1 章图 1-14）。组成一个系统的不同组成部分也可以有不同的产品生命周期。例如，电子技术持续的技术进步使得汽车的产品生命周期相当多地超出了电子控制元件（ECU）硬件和软件的生命周期或变化周期。另外，在开发、生产和运行阶段中的系统需求也是不同的。

举例：在开发阶段、生产阶段和服务阶段，对电子控制单元（ECU）接口的不同需求

在开发、生产和服务的不同阶段，在生命周期的单一阶段，仅仅通过引入不同硬件特性和软件功能，就经常能满足电子控制元件接口的不同需求。图 3-6 描述了开发阶段、生产阶段、运行和服务阶段之间的一些功能区别。此外，对于系统接口的数据传输能力的要求也是不同的。

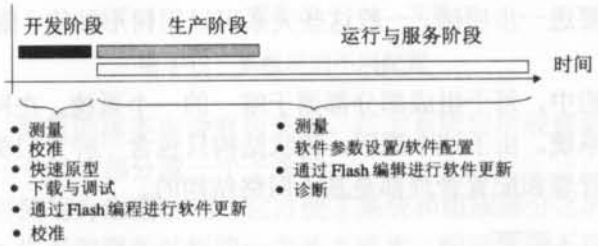


图 3-6　对电子控制单元接口的不同需求

3.3.2　型号系列

不断增加的汽车型号和上涨的消费者指数（也就是关于一些个性和扩展性的可用选项）相结合导致了车载系统需要较多的型号系列数。满足这些系统需求的方法可以引入另外的组成部分型号，或引入可扩展组成部分的系统结构。图 3-7 中，型号范围被组成部分 X 扩展了，而图 3-8 中通过添加组成部分 Z 也得到了同样的结果。

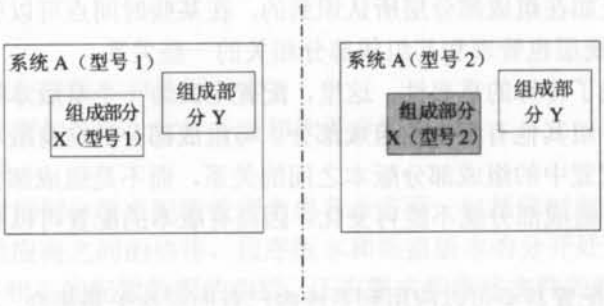


图 3-7　根据组成部分的型号创建系统型号

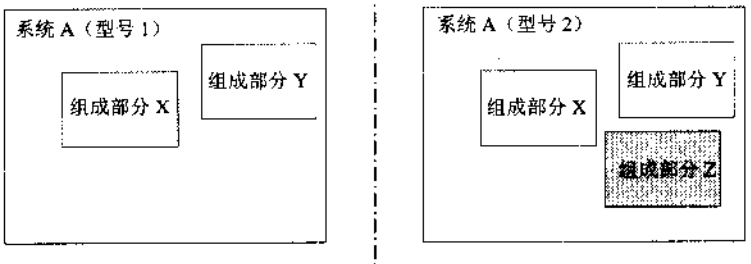


图 3-8 通过扩展组成部分创建系统型号

3.3.3 版本和配置

因为系统型号可能存在于所有的系统层，所以在系统、子系统、和/或各组成部分之间的层次关系应该需要进一步明确。一般这些关系可以用树形结构（图 3-9）或网络结构（图 3-10）进行描述。

在一个树状结构中，每个组成部分都属于唯一的一个系统。在网络结构中，一个组成部分可以属于几个系统。由于这个原因，树状结构只包含一种专门类型的网络结构，而处理系统型号的版本管理和配置管理都是基于网络结构的。

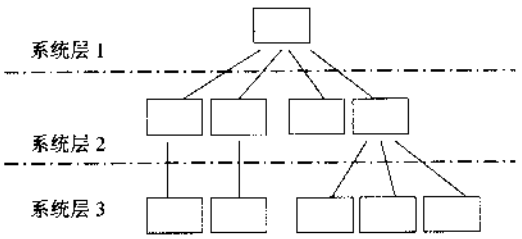


图 3-9 树状结构

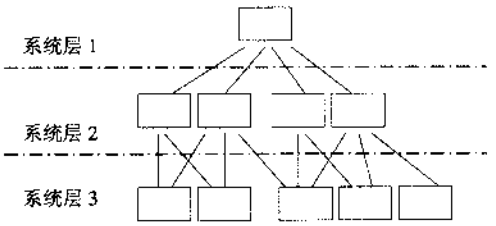


图 3-10 网络结构

随着时间的演化，系统的进一步开发和汽车生产阶段新系统的引入会导致新一代系统组成部分的开发。正如在组成部分层所认识到的，在某些时间点可以引入所谓的组成部分版本（图 3-11）。系统层也管理和其组成部分相关的一些关系。

因此，配置呈现了特殊的重要性。这里，配置定义为一个有版本的组成部分，这个组成部分依次参考了一组其他有版本的组成部分。与组成部分的自身版本不同，配置所管理的仅仅是与包含在配置中的组成部分版本之间的关系，而不是组成部分版本自身。一旦组成部分的版本确定，组成部分就不能再变化。因而有版本的配置可以参考仅仅有版本的组成部分。

基于这些定义，配置甚至可以应用到系统内已有的层次关系集合。在这种情况下，配置“仅仅”包含层次关系。然而，值得提出的是层次关系也可能进一步地开发或变化（图 3-12）。

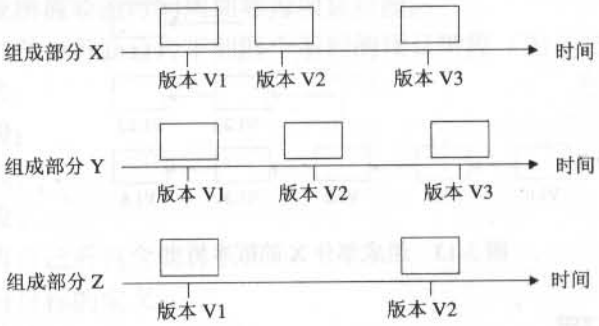


图 3-11　组件层的不同版本

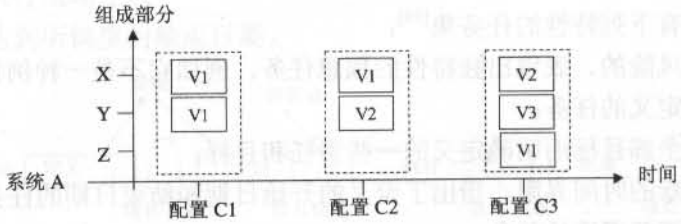


图 3-12　系统层的不同配置

随着时间的演化，配置的版本也会有所变化。在主系统层所观察到的这些后代版本可以用相同的方式看作一个组成部分或一个子系统。

版本和配置管理可简写为配置管理，它方便了系统和组成部分之间已有关系的管理。这使得它成为了开发、生产和服务过程的一个基本要素。配置管理不仅提供了型号的并行开发，还提供了连续版本的开发，以及在所有系统层保证了产品生命周期在一些个别阶段不同系统需求的满足。

在开发过程中，配置管理覆盖了文件归档、文件管理、文件修复以及所产生的结果交换中所有必要的过程步骤。配置管理还包括了所有过程阶段中的信息交换，其中不仅包括了公司内不同组织机构之间的信息交换，而且还包括了不同的开发合作者之间的信息交换。除了和工作相关的决定和结果之外，配置管理也管理所有的原料和工具，例如配置的开发工具。这是确保过程步骤的重现性或重复性的唯一方法。

配置管理所管理的几个条目如下所列：

（1）需求；

（2）规格说明；

（3）实现（例如程序版本和数据版本）；

（4）描述文件（例如诊断、软件升级和软件参数表示）；

（5）文档，等等。

尤其在软件开发期间，要求配置管理考虑各个方面，包括同时发生的即并行的工作进程，汽车制造商和供应商之间的协作，程序版本和数据版本的分开处理，历史版本以及与软件组件（图 3-13）相关的配置数据的跟踪，还有需求和描述文件的管理。由于这些原因，开发过程中一个重要的先决条件是对配置管理进行方法集成。

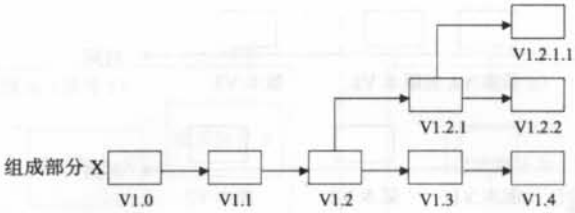


图 3-13 组成部分 X 的版本历史

3.4 项目管理

项目描述了具有下列特性的任务集[69]：

（1）包含内在风险的、表现出独特性的预想任务，例如它不是一种例行公事的情形；

（2）具有明确定义的任务；

（3）为了达到全部目标而明确定义的一些责任和目标；

（4）有一个清晰的时间界限，指出了定义的开始日期和结束日期的任务；

（5）有受限的资源调度的任务；

（6）调整组织结构以满足规划目标的任务；

（7）许多情况下，存在一些可辨别的和有区别的做出贡献的任务或项目阶段，以及一些相互连接和相互依赖的组织单元。

特定项目的目标可用下列目标形式表示（图 3-14）：

（1）质量目标：全部的项目结果要满足的需求是什么？

（2）成本目标：要达到全部的结果，经核准的预算是多少？

（3）目标期限（或里程表）：所期望的全部结果什么时候可以表达？



图 3-14 项目目标

因为任何开发所作的努力都可以用上述某几个特性表示出来，所以它也可以被视为一个专用的项目。

一方面，项目管理包含了项目计划的所有方面，而项目计划本身关心项目目标的执行。所以项目管理有单独的质量、成本和最终期限计划，也有项目调度、并行的组织计划、人员部署调度和风险分析。

另一方面，项目管理也包括项目控制和项目跟踪的各方面，例如从项目执行一直到项目尾声的全过程中，质量、成本和最终期限的跟踪和监控。它也监视一些迄今可能一直隐藏的或未被承认的风险。如果发现了这样的危害，那么相信风险管理可以应用一些必要的过程进行修正。

3.4.1 项目计划

开始时，必须定义一些可单独组成项目的任务。有时通向项目结束的艰难过程存在一些转折点，也就是说里程碑式的或单独的项目阶段结束的一些重要事件。这样一个转折点

很可能是已调度的部分出货的交货日期，以及测试程序或消费者进行支付等事件。完成一个单独的项目阶段所需要的时间周期称为项目阶段。

通常项目计划至少可以分为下列四个不同的项目阶段（图 3-15）：

（1）定义阶段；

（2）计划阶段；

（3）执行阶段；

（4）完成阶段。

每个阶段都有自己所包含的转折点：

（1）得出项目目标的定义；

（2）项目计划完成；

（3）项目执行成功完成；

（4）项目达到所调度的结束日期。

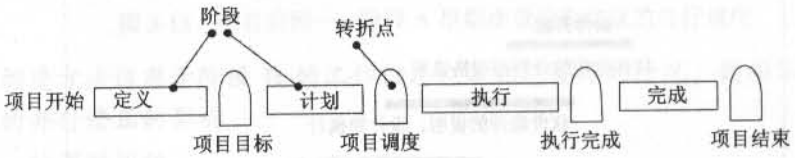


图 3-15　项目阶段和转折点

一些情况下，上述四个主要的项目阶段可以再分为另外的一些阶段。尤其在项目中有几个组织机构和一些不同的协作企业进行合作时会发生这样的情况。在汽车开发中也时常有这样的情况。

3.4.1.1　质量计划

在项目管理的整个过程，质量计划定义了所有的措施，从而确保全部的项目结果都满足所设计的项目需求。在所有的项目阶段，定义质量保证指导方针和质量测试措施或方法。

3.4.1.2　成本计划

成本计划包含全部的资源计划，以及为完成项目所需要的经费支出。这里最常使用的一些方法包括人员部署调度和资金支出调度。成本计划也需要考虑一些可能的节省成本的措施，例如再使用以前项目的一些资金。

3.4.1.3　时间计划

项目调度决定了完成每个项目阶段所允许的时间。它是一个给每个阶段和转折点分配明确的开始日期和结束日期的过程。除了可能的不同项目的人员部署分配之外，项目调度要关键考虑的是处理并行项目的结果。还有，如果人员不够，另外需要考虑项目阶段之间的相互依赖，从而把项目调度变为一个高风险的调度行动。

举例：项目调度——汽车电子系统的开发

图 3-16 显示了来自于汽车电子系统开发项目的一部分项目调度。整个汽车的最终期限

调度必须和所有项目阶段所分配的调度是同等的和同步的。这个例子中，汽车电子系统软件的开发调度必须进行调整以产生所需要的相符合的调度。

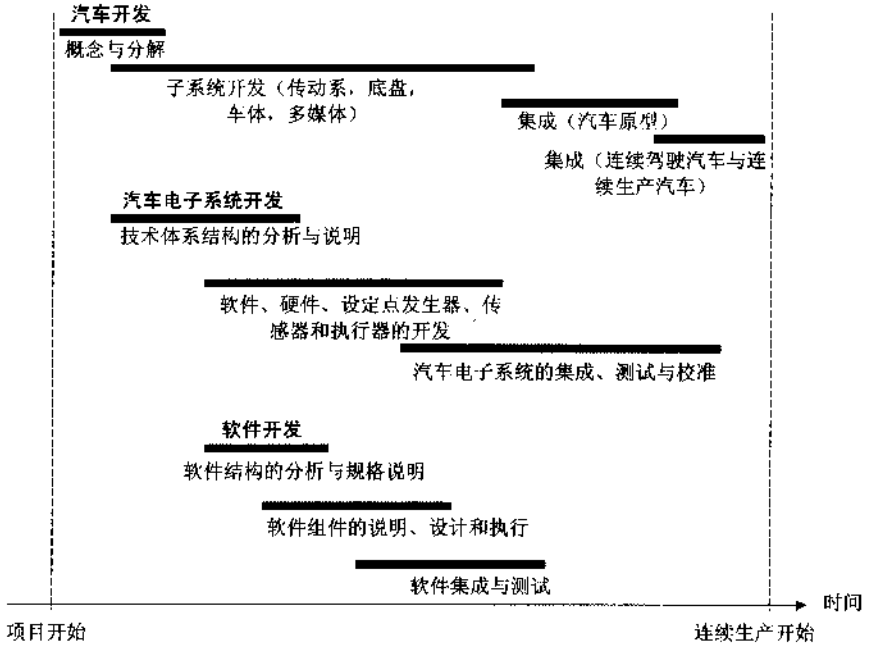


图 3-16　汽车系统开发的时间计划

如图 3-16 所示，可以顺序调度几个任务，但是其他任务必须同时执行。许多情况下，对项目任务的并行处理进行计划就能削减开发时间。由于这个原因，并行开发步骤的调度和同步，称为并行工程，是一个主要的挑战。通常属于不同团队的大量雇员可以处理这个基本功能。越过汽车制造商、供应商和其他开发合作者的合作边界，这个基本功能也经常被共享。

沿着信息流通道以及同步点的相互连接处，一个过程模型可处理这项正确定义单一任务的工作。因而这项工作成为了成功的项目管理的一个基本的先决条件。软件开发和汽车电子系统之间的复杂配合，以及汽车开发，都与其他工业应用领域（例如长途通信）所采用的普遍方法形成了显著对照。因此可以说汽车系统工程自然地对汽车工业的软件工程造成了重大的影响。

至于显示相互依赖的两个项目阶段，其基于时间的同步在下面所列的三种情况之间可能存在区别。

情况 1：阶段 B 在阶段 A 结束后开始（图 3-17）。

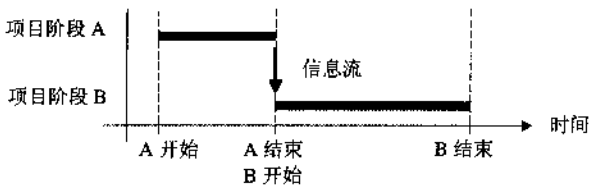


图 3-17　项目阶段——顺序调度

这种调度方法适用于阶段 A 的完成作为阶段 B 开始的前提，或阶段 B 的处理大部分依赖于阶段 A 所产生的结果的情形。

优点：可以没有内在风险地提供连续的信息流。

缺点：需要的处理时间长。

例如，控制单元软件（阶段 B）集成测试的开始必须等到集成软件（阶段 A）的结束，才能开始进行。

情况 2：阶段 B 在获得阶段 A 的部分信息后，即阶段 A 所采用的早期决策没有结束之前开始（图 3-18）。



图 3-18　项目阶段——阶段 A 早期决策没有结束的并行调度

这种调度方法适用于阶段 B 的工作信息包是相对鲁棒的情况，因而不受阶段 A 原始决策所作的并行修正的影响。

优点：处理时间短。

缺点：阶段 B 的重复造成了延迟风险。

例如，使用快速原型系统，应用软件（阶段 B）的功能开发可以在平台软件（阶段 A）完成之前开始。

情况 3：阶段 B 在阶段 A 的部分信息即阶段 A 的早期决策结束时开始（图 3-19）。

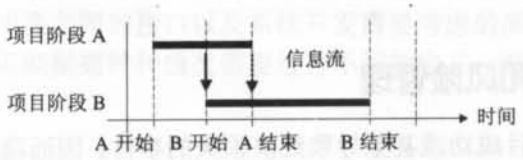


图 3-19　项目阶段——阶段 A 早期决策结束的并行调度

在阶段 A 所作的决策没有快速形成最终结果的情况下，也就是说，在所作的决策越接近时间线的开始，也就越接近达到它们的最终状态，而且并行变化的风险是相对低的情况下，这种调度方法是具有优势的。

优点：处理时间较短。

缺点：由于受阶段 A 的一些早期约束，所以存在质量损失风险。

例如，甚至在应用软件的所有功能明确和实现（阶段 A）之前，那些已经实现的功能就准备校准（阶段 B），虽然朝着阶段 A 时间线的末端，但是发生的修正的可能性会很低。

在上面描述的三种情况中，交换的信息必须映射为开发过程。定义这些过程的任务在本书第 4 章会进行更为详细地讨论。

3.4.1.4　开发角色和任务

就像前面第 2 章所指出的，汽车功能的开发需要不同学科领域的专业人员的参与。因

此，开发团队是各学科的专业人员构成的。单个团队的成员们贡献他们各自的才能，他们可以假设为各种角色。

关于任务履行和职责要点，需要根据定义清楚地划分所有做出贡献的角色。表 3-1 给出了开发项目中经常出现的不同情况下不同角色的综述。

表 3-1 开发过程的角色和任务

角　色	任　务
功能开发者	分析用户需求，详述逻辑体系结构
系统开发者	分析逻辑体系结构，详述技术体系结构
软件开发者	分析软件需求，以及软件说明、软件设计、软件实现和软件测试
硬件开发者	分析硬件需求，以及硬件说明、硬件设计、硬件物理实现、硬件集成和硬件测试
设定点发生器、传感器和执行器开发者	分析设定点发生器、传感器和执行器的明确需求，以及它们的说明、设计、物理执行、集成和测试
集成、测试和校准者	外部车辆系统及其功能的集成、测试和校准

汽车电子系统的软件功能开发需要表 3-1 中所有角色的参与。至于作为结果的角色分配需要克服各学科间的频繁交叉合作的障碍。由于这个特定的原因，专注于软件功能开发的各学科间的联合作业最好基于图形功能模型进行。在一定程度上，越来越多的以前所采用的平淡无奇的规格说明被替换掉。

在制定任何一个项目的计划时，必须考虑所涉及的角色分配以及团队成员的不同才能。

汽车工业本身所特有的另一个特色是具有不同的开发环境。如图 1-18 所示，通过仿真方法，使用基于实验室的开发程序，在测试平台或测功机上，以及在汽车中必须把虚拟的步骤进行同步。

3.4.2 项目跟踪和风险管理

风险包含可能危害项目成功或甚至导致经济损失的事件。因而项目风险可能由质量风险、成本风险或最终期限风险组成。

风险管理包含了处理和项目相关的风险的所有措施。因为它和项目控制紧紧交织在一起，所以风险管理会对事件搭配一些合适的对策。例如在一个项目开发的过程中，与实际的目标相对照，会显示确定的目标偏离。而作为防范风险的一个措施，功能开发可以帮助本身重现以前的有效功能。另外用开发原型可以进行新功能的早期确认。尽管提出了这些方法，但还是没有一种准确无错的可以规避整个项目风险的方法。所以风险的存在使得项目计划的修缮时常成为必要。

3.5 供应商管理

汽车工业中汽车电子系统的开发，在汽车制造商和供应商之间，经常表现出显著的差异。汽车制造商通常定义要开发的用户功能需求，而所期望的功能依靠汽车电子系统实现，而这经常是供应商的独占领域。汽车制造商通常负责微调和汽车功能的验收测试。

3.5.1　系统及其组成部分的责任承担者

面对这种显著差异的特定风险时，汽车制造商和供应商之间的精确定义就成了一个必不可少的先决条件。图 3-20 中，利用 V 模型图可以恰当地对汽车制造商和供应商进行形象化表示。由于汽车制造商对汽车负有责任，所以汽车制造商在 V 模型的左边分支的子系统开发中，以及右边分支的汽车开发中，而供应商经常负责组成部分层。

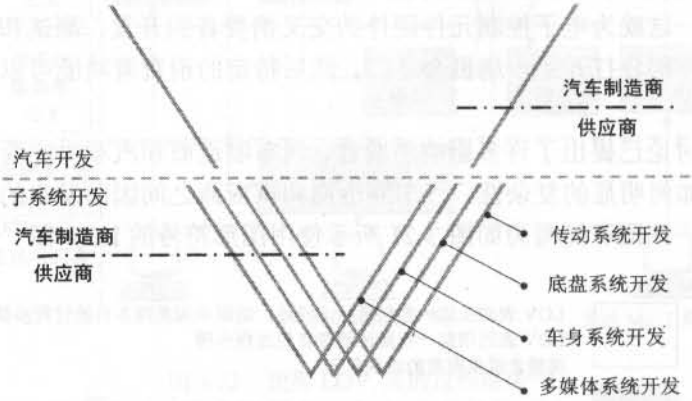


图 3-20　汽车制造商和供应商之间的职责划分

除了技术方面的澄清之外，越过合作边界的合作也需要澄清项目的组织层和法定层所提出的所有问题。由于这个原因，承包商管理在汽车管理中是一个非常重要的领域。它包含了汽车制造商和供应商之间的接口以及系统开发需要考虑的所有任务。

虽然涉及的接口可能根据种种情况需要进行不同的定义，但是对每一种情况它们都需要正确且完全的定义。

3.5.2　规格说明和集成的接口

汽车制造商和供应商之间的合作可在两类接口之间加以区别，如图 3-20 所示。

（1）规格说明接口在 V 模型的左边分支；

（2）集成接口在 V 模型的右边分支。

这些接口可能会变得非常复杂。根据大量现有的相互关系，已证明可以简化这些接口。例如，如果一个系统包含 n 个组成部分，即 n 个不同供应商，那么在系统开发的主办方中必须控制汽车制造商这一边，以及规格说明方和集成方两边 1 到 n 个关系。现在，我们可以保证供应商会给 m 个不同的汽车制造商提供一个特定的组成部分。这就意味着，在负责组成部分开发的主办方中，必须再控制组成部分制造商这一边，以及规格说明方和集成方两边 m 到 1 个关系。

3.5.3　开发过程的交叉合作定义

尽管规格说明和集成的接口复杂，但是这种情况给供应商和汽车制造商都提供了许多

优点。

多数情况下，汽车电子系统包括嵌入式系统，那就是说，在一定情形下集成的系统对各个用户来说不会公然显示他们的存在。通常电子控制元件的价值深藏在他们所实现的功能中。许多情况下，电子控制元件的功能就表示了不同的竞争系数。

由于这个原因，汽车制造商真正感兴趣的是那些具有内在特性的处在竞争边缘的电子控制元件功能。对软件开发者来说，这意味着软件消费者的注意点就是集中在那些具有适当竞争希望的电子控制元件应用软件的功能上。

对供应商来说，这就为电子控制元件硬件的交叉消费者的开发、测试和标准化，平台软件，以及应用软件部分打开了一扇机会之门。然后特定的消费者功能可以被添加到和集成到应用软件中。

前面几部分的讨论已提出了许多影响消费者、汽车制造商和汽车供应商之间合作的因素。不管他们具有如何明显的复杂性，汽车制造商和供应商之间因而发生的相互关系可用图的形式进行描述。该图可简写为如图 3-21 所示使用图形符号的 LOV 图[23]。

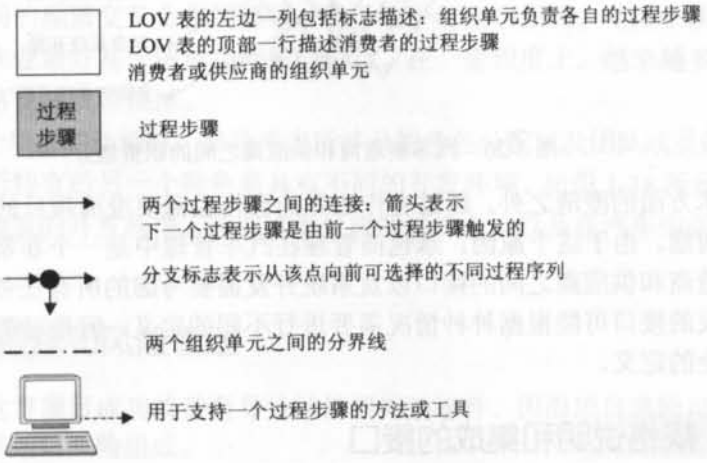


图 3-21 LOV 图中所使用的过程描述符号[23]

图 3-22 显示了 LOV 图的一个例子。顶行描述了汽车制造商保证负责的一些过程步骤，这里汽车制造商的功能是担当消费者。沿着分配给他们的过程步骤，下面几行表示供应商的组织机构，连接线和箭头表示了过程步骤的顺序。每一个箭头表示中间结果流，称之为人造物品。为了不同的过程步骤，所采用的方法和工具的定义可表示为独立的一行。

3.6 需求管理

很自然地，不必为了汽车的应用对需求管理进行专门设计。需求管理的一个显著特点是它和配置管理是具有共同点的。这就说明了汽车相关项目在处理需求管理时所采用的都是标准工具。然而应该注意的是，在汽车项目中必须考虑特定需求，例如，在需求管理中，

支持跨企业、跨地域的交叉合作，需求管理和配置管理的相互作用等。所有上面提及的需求都强调了需求管理必须作为一个整体有系统地集成到一个开发过程中。

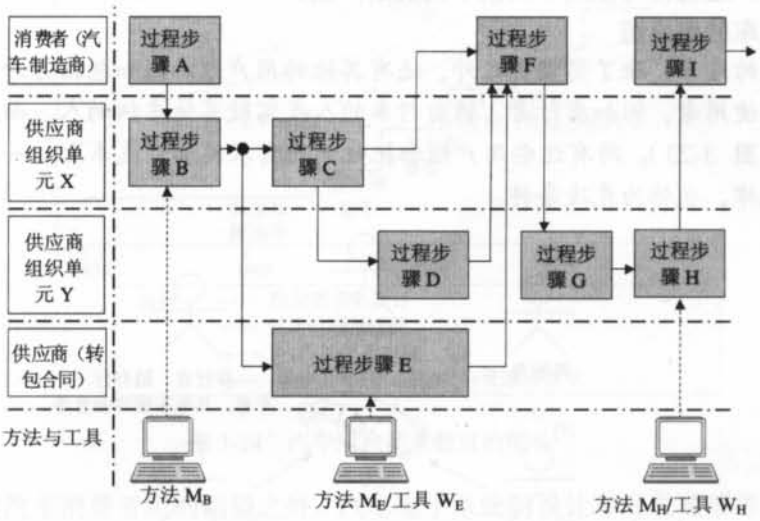


图 3-22　使用 LOV 图的过程描述[13]

就像它的名字所暗示的，需求管理关心和自身有关的所有任务，包括：

（1）需求注销和记录；

（2）需求跟踪。

然而，有必要注意的是需求分析以及给逻辑系统结构和技术系统结构定义规格说明的任务都是属于系统的核心过程和软件开发阶段的。

3.6.1　用户需求挖掘、记录和解释

任何一个产品都会被满怀着接受和成功的希望送到市场，所以它必须满足其目标用户组的需求。由于这个原因，首先必须尽可能正确地定义一个特定系统的有意向的用户及其期望。因此未来用户需要的和想要的就称为用户需求。因为部分用户可能不具备技术背景知识，所以必须采用用户最熟悉的语言描述用户需求。

这样用户的希望和期望成为了下面所有开发步骤的驱动。识别、记录和解释用户需求的过程步骤断然不同于所有的并行开发步骤，该步骤的执行应该采用的方式是将透彻的研究和高度的相互作用相结合。甚至给定这样的事实，即目标组所表示的不是所有的需求，但看起来可能是实际的或可行的。仅仅如果为了从总体上得到更好地对用户期望的处理，这样的事实更应该被记录和评价。为了这个原因，多数情况下用户需求是可用的，仅以各种各样的列表条目形式就可产生各种结构化的基本表格。

基于他们所获得的时间点，可以分为下面三类用户需求：

（1）项目开始时所表示的需求；

（2）在项目开发过程中所提出的通常涉及要求改变的或追加的需求；

（3）在成品运输之后，通过反馈方法返回的需求，通常称为新的需求、故障报告或改进建议。

在以下的讨论中，所有列出的需求都被给予了同等的待遇。

用户一词是用于和完成的系统有关的一个或全部的人员，并把他们的愿望和/或指令对系统施加影响。这通常可以用于辨别不同的用户组。

举例：汽车的用户组

对于特定的汽车，除了驾驶员之外，还有其他的用户组。例如包括另外的用户，不仅有其他的道路使用者，例如步行者、骑自行车的人或驾驶其他车辆的人，而且还有服务人员和立法者（图 3-23）。所有这些用户组都把他们的需求施加到汽车上。一些需求种类，例如法定的规律，也称为先决条件。

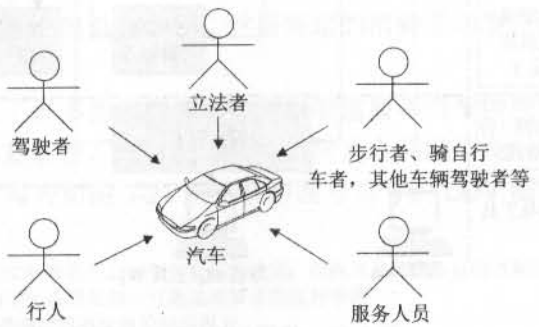


图 3-23　识别特定汽车的用户组

在称之为建议挖掘阶段期间，需求和解决方案之间的差异显示了许多优点。用户通常倾向于通过陈述或建议方案的形式表达他们的需求。如果一个用户为解决方案提供了建议，那么应当明确表达这些建议，从而使得所建议的方案可以用于明确地描述潜在用户的需求。如果正确的挖掘失败，那么在不合适的早期就可以采用技术实现，从而限制解决方案的可用空间。

举例：关于油量指示的用户需求表示

一个用户表示了以下的建议方案：燃油量应该用以升为单位的确切值来代替$\frac{1}{4}$、$\frac{1}{2}$、$\frac{3}{4}$、1 这样的比例指示。

潜在的用户需求可以为：汽车可续驶的里程范围的表示应该更为精确，即精确到码或千米。

可以用不同方法挖掘上述用户需求。首先想到的是根据已有系统和要求的改变经演绎推理验证的采访和研讨会。还有用户从场所反馈的潜在的固有性。应该想到用户需求可能来自于大量背景，例如技术的、组织的或经济的自然状态。

用户需求经常可基于不同的标准进行分类，例如来源、优先级、紧急性、稳定性、易测性、接受程度和相似性。

在过去的十年里，对汽车发展的进一步观察揭示了大量用户需求是随着每一个连续的汽车年代增长的。可以说这种趋势不会随着汽车功能数的增多而独自增长。这相当大的贡献也来自于汽车型号数的增多和消费者期望的上升，以及已经成为主要因素的汽车消费和汽车系列（图 3-24）[9]。

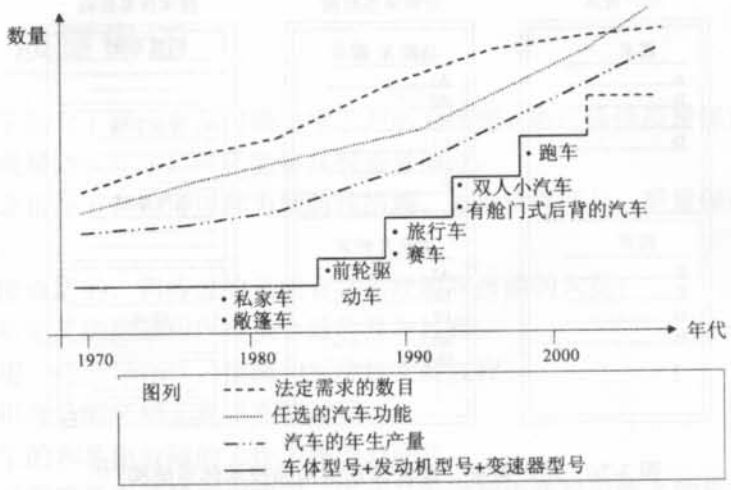


图 3-24　汽车用户需求数目的增长[9]

除了特定汽车消费者组的期望之外，汽车电子系统的设计和开发需要考虑大量附加的先决条件，例如一些技术规格说明和规定的约束。图 3-25 略述了需求种类。这些种类在多数情况下必须都是可观察到的，在设计汽车电子系统时也是必须要考虑的。



图 3-25　电子系统的不同需求种类

汽车电子系统上多种形式的需求显示了众多的相互依赖以及互惠的作用。需求也可能是相互对立的。如果是这种情况，在技术实现成为可能之前，必须解决这种导致特殊目标的冲突。

接受的用户需求成为了所有并行开发步骤的基础。在任何系统的开发中，系统需求用多方参加的项目学科语言描述。因为这个原因，用户需求和那些从开发的角度施加于系统的需求之间需要进行区分。本书第 4 章还引入了另一种逻辑体系结构和技术体系结构之间的区别。为了把用户需求映射为技术体系结构，第 4 章同样也讨论了所要求的分析和规格说明程序。因此，为了方便计划和跟踪需求的执行，除了用户需求观点之外，需求管理还必须支持逻辑的和技术的体系结构观点（图 3-26）。



图 3-26 用户需求、逻辑体系结构和技术体系结构[12]

3.6.2 用户需求跟踪

处理系统型号、系列和部署的组成部分的不同生命周期，以及各种汽车中组成部分的重新使用，所有这些挑战都成为了系统设计阶段所必须考虑的先决条件。同时，这些挑战经常导致了不同汽车项目之间的横向联系。在项目计划中，必须以同样的程度把这些联系考虑为汽车制造商和供应商之间的众多的相互关系，不涉及设计的相关任务的并行过程，或并行工程。

因此，值得注意的是，除了用户需求的挖掘、记录和解释之外，一个最重要的需求管理功能还包括需求执行的跟踪。

为了有利于开发的全部实体，在决定用哪个程序和/或数据版本执行需求，或者不用程序和/或数据版本执行需求中，需求跟踪是唯一常用的方法。在中间版本的集成和质量保证中，需求跟踪也是一个非常重要的需求。作为需求跟踪的先决条件，必须管理用户需求、逻辑体系结构和技术体系结构之间的相互关系。最后，如图 3-27 所示，所有系统的组成部分必须连接成各自的需求。

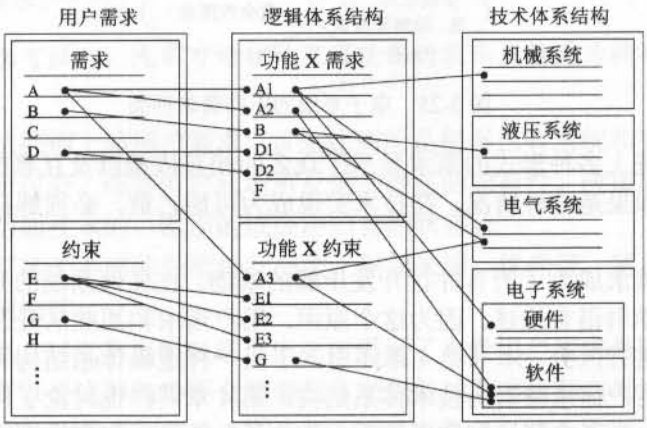


图 3-27 需求跟踪[12]

3.7　质量保证

质量保证包含了确保产品可满足需求的所有措施。通过提供质量保证指导方针以及已建立的合适质量测试程序，产品能够具有质量保证。

质量保证指导方针也可以称为预防性措施。关于软件产品，质量保证指导方针包括以下一些措施：

（1）调度合适的、训练过的且拥有必需经验和技能的人员；

（2）用所定义的测试程序提供合适的开发过程；

（3）采用一些指导方针、措施和标准以支持过程；

（4）提供合适的环境工具以支持过程；

（5）手工的和易出故障的工作步骤的自动化。

质量测试程序是为了检测出错误。因此开发过程中在尽可能多的步骤之后，应该执行质量测试。因为 V 模型中集成了多种质量测试方法，所以本书第 4 章将和核心过程一同更为详细地讨论这些方法。

在软件产品中，一般可分为下面两种错误：

（1）规格说明错误；

（2）实现错误。

研究显示了在多数项目中规格说明错误是重要的。由于这个原因，V 模型区分了正确性验证和有效性验证。

3.7.1　集成和测试程序

一旦明确地定义了特定项目的用户需求，就有可能通过用户需求来测试产品。V 模型分为下列四个不同的测试步骤（图 1-15）：

（1）对照组成部分的规格说明，进行组件测试。

（2）对照技术体系结构的规格说明，进行集成测试。

（3）对照逻辑体系结构的规格说明，进行系统测试。

（4）对照用户需求，进行验收测试。

组件测试、集成测试和系统测试都包含了一些正确性验证措施。验收测试是一个有效性确认测试。

测试是检验故障存在的一种方法。它们集中于故障识别，并有助于达到相同的产品质量。应该在所有系统层尽可能早的连接处进行测试。然而这里应该注意的是，一旦用于识别故障的测试失败，那么并不表示故障必然不存在。因而，测试必须与另外的质量保证措施一同计划。

根据应用程序在设计阶段所定义的一些相关测试案例，所执行的测试可以被表示出来。这也意味着用户需求隐含定义了一些接受测试。系统测试的应用案例被定义为逻辑体系结构的规格说明的一部分，而集成测试的样本案例成为了技术系统结构的规格说明的一部分。

3.7.2 软件质量保证方法

软件质量保证所用的方法是和集成程序息息相关的。尤其考虑到日益增长的和许多基于软件的汽车功能相关的安全性，如今质量保证的重要性也稳步增长。有一个简单的方法可用于明确表达安全性需求，即特定的汽车功能所需要的所有组成部分，以及所有辅助系统，都必须满足技术安全需求。

在汽车电子系统开发中，着眼于所涉及的硬件不可能简单地达到高度的系统安全。因此软件质量测试方法的应用是完全必要的。

由于前面几部分所讨论的种种原因，软件可靠性和安全性的验证越来越受到重视。实际上，验证方法已经不可能保证从用户需求的分析到程序的完成可以无故障执行。因此，创建可靠软件唯一安全的途径就是不断应用软件质量保证方法和不危及安全的组织方法，例如需求管理和配置管理。它结合了上述方法，以及便于开发过程管理的软件项目方法的调度。

图 3-28[70]显示了一些可能的软件质量保证方法，包括正确性验证和有效性验证。本书第 5 章还会更为详细地讨论其中的一些方法。

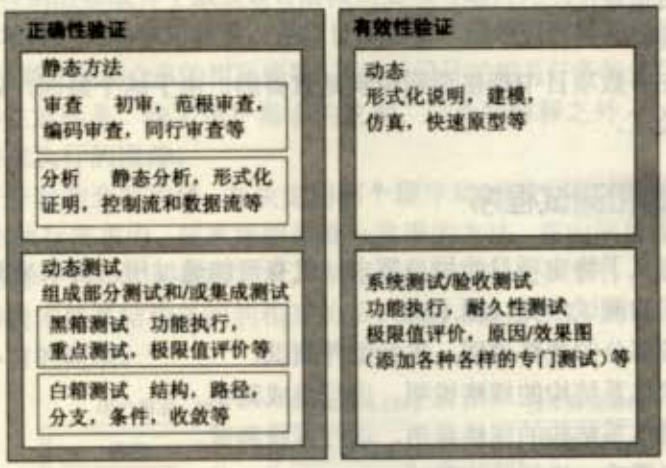


图 3-28 软件质量保证方法综述[70]

第 4 章 Chapter 4

电子系统和软件的核心开发过程

与组件的开发相比较（组件开发是单个组件的分析和设计），系统工程是把系统的设计和分析当作一个整体来考虑。根据文献[12]和[71]中的定义，系统工程的定义如下：

系统工程是以科学和技术资源的专门应用为目的：

（1）把系统的功能需求变成系统结构的描述，最大可能地考虑到所有的功能与效率要求。

（2）集成所有的技术参量，确保所有物理、功能和技术接口的兼容，整个系统的定义和设计尽可能地达到最优。

（3）整个系统的开发集成各个专业学科的贡献。

系统工程是一门交叉学科，包括所采用的一系列措施和方法，其唯一目的是推动特定系统的成功实现。在系统工程中，开发过程首先是完成系统功能需求的定义，紧接着是系统的设计、鉴定和验收。开发过程涵盖了系统的整个周期，包括开发、性能特征、成本、项目时间表、测试、制造、运行、服务、培训和系统处理等各个方面。

系统工程提供了一套结构化的开发过程，考虑到产品生命周期的所有阶段，从最初的概念到产品，再到操作和服务。系统工程必须考虑技术和组织方面。例如，开发汽车内应用的电子系统，需要结合本书第 2 章介绍的工程学科内容。

与系统工程相比，软件开发与设定点发生器、传感器和执行器等硬件开发一样，是整个系统开发过程中的一门专业工程学科分支。

明确地定义系统工程和软件工程间的功能描述和集成的接口是开发过程不可缺少的先决条件。这章将重点讨论电子系统的核心开发过程。下面章节的讨论主要依据图 4-1 描述开发过程和内容来展开。

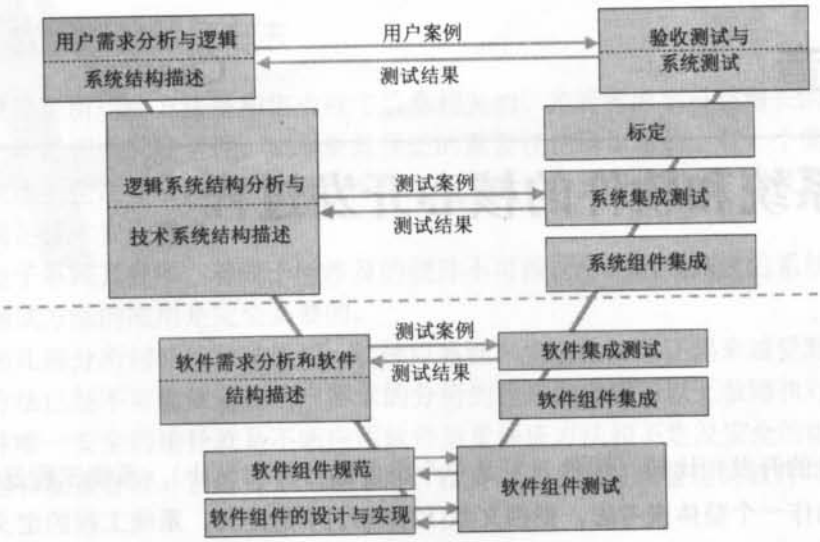


图 4-1 电子系统和软件的核心开发过程

4.1 要求和限制条件

4.1.1 系统和组件责任的分担

在汽车开发中，系统和组件的开发经常由多个开发团队分工完成。如果供应商相互间处于竞争环境中，这将影响合作的途径和方法。因而，在这样的状况下，汽车制造商通常负责整个系统的开发，而供应商负责子系统和零部件的开发（见图 4-2）。

因而，汽车系统的规格描述、集成和质量保证等相关功能通常由多方一起参与完成，他们分别承担汽车制造商、系统供应商或者组件供应商中的任务之一。

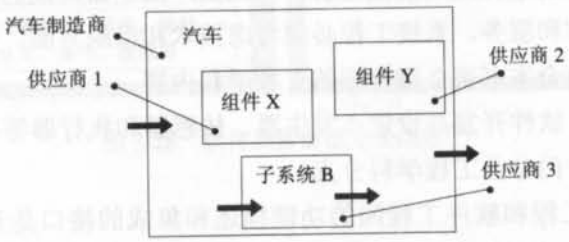


图 4-2 组件、子系统和系统开发的责任分担

4.1.2 系统工程和软件工程的协调

如图 4-1 所示的开发过程模型中，早期和后期开发步骤的目的是评估与汽车整体系统相关的功能与性能方面。开发过程中的重点在于系统的各组件间的相互作用，并由此实现汽车系统的功能。单个看，系统组件可以通过不同的技术方式来实现，如基于机械、液压、

电气甚至电子系统来实现。然而，只有系统组件相互间具有良好的协调作用才能达到用户所期望的汽车功能。

车内 ECU 与多种不同的组件之间存在相互联系和相互作用。汽车功能的设计就是在具体的技术系统上构建系统逻辑功能。技术系统包括 ECU、设定点发生器、传感器和执行器。为此，系统和其组件间的接口必须明确定义，并且建立系统的子功能与组件间的联系。

本书所讨论的组件或子系统限于用软件技术来实现的子系统或组件。这个过程需要充分考虑系统和软件开发之间的相互影响，因而，软件的开发必须应用系统化的开发方法论才能进行，这就是所谓的软件工程。实际上，软件工程不是一个新的发明。很长一段时间以来，几乎在所有的应用领域，许多有经验的软件开发人员应用的某些原理，大部分或多或少地受这些还没有正式定义的软件工程中的一些原理的影响。

协调过程最重要的先决条件是系统开发与软件开发间接口的详细定义。因此，下面的部分将特别注意系统和软件工程间的集成接口。

从 4.5 节到 4.10 节，软件作为组件或子系统占有中心作用。更详细地分析，这些子功能由微处理器上的 ECU 执行并且基于软件来描述。因为软件可以实现任意的逻辑和数学关系，可以计算出给定函数对应输入信号的输出变量，软件功能占整个汽车功能中的比例在不断地提高。同时软件的复杂性也在增加。如何可靠地处理和解决软件的复杂性问题是一种挑战，这个挑战需要利用合适的软件工程方法，例如软件标准接口的应用。

如同讨论表明的，软件工程指明了把逻辑系统结构映射到具体软件系统的方法，即映射到整个程序和由车内分布式处理器控制处理的系统数据。此外，在第 3 章中所描述的一般性的支持过程在软件开发过程中的应用也是非常有益的。

下面各节讨论系列生产的汽车 ECU 软件的功能描述、设计、实现、测试和集成的限制条件。特别强调的是，软件逻辑功能的描述与针对特定微控制器的程序和数据的设计和实现之间的明确区分。

程序版本和数据版本分离极大地简化了开发、生产和服务中的版本管理工作。这点也需要适当的考虑。

V 模型为嵌入式系统而开发。把软件看作信息技术系统的组件，对软件和硬件开发给予同样的重视，并且集成了质量保证测试。

在可靠性和安全性方面具有较高要求（规定了相应的测试步骤）的系统开发，以及系统的组件分布式开发的系统，V 模型是他们优先的选择。

在如此多的有利条件下，V 模型也有一些不利因素。一个缺点是在早期开发阶段缺乏反馈机制，使得在早期阶段的错误或者改变只有在很晚的阶段才识别出来或考虑到。用这个方法，新的要求和计划改变只有在花费很大的情况下才能调整和/或者执行。事实上，他们又有引入不安全结果的倾向，增加了方案风险甚至威胁整个方案。实践结果表明，V 模型是通过重复开发过程来循环的。在图 4-1 中各种可能的开发过程的重复循环是通过测试结果来决定的。在很多情况下，第一步是建立一个原型，尽管功能有限，原型能在开始的时候用来验证现实世界的环境。这有助于鉴定早期缺陷。然后，有限原型通过循环来确定开发的改进原型。一直重复这个过程，直到所有的要求或质量目标都满足。另外，即使在时间紧迫和成本有限的情况下，有限功能的软件版本也能被装载运行。这种增量和迭代方

法，在软件工程中可以及时减少开发风险。用来描述这个迭代过程的其他术语是原型模型或者螺旋开发过程模型。

V 模型也应用在汽车工业中的系统开发过程。首先开发完成系统的组件，然后在实验车中进行测试，此后进一步建立汽车原型和在道路上的真实汽车中进行现场测试。随着迭代次数的增加，开发的风险和每一个循环步骤成本降低。每个集成版本开发的原型也称为样品。根据项目的进展，经常区分为 A、B、C 样品。系列生产版本也叫做 D 样品。

在软件功能的开发中，类似的方法是通过应用仿真和快速原型工具来支持，更多的细节描述将在第 5 章的 5.3 节讨论。

4.1.3 基于模型的软件开发

软件模型有利于设计没有解释误差的算法。这是通过术语学和图解来完成的，术语学和图解比纯文本和程序代码表达得更为清楚。

因为这些明显的优点，在图解的基础上基于模型的软件开发在最近几年的汽车工业获得广泛的认同。各种软件模型描述了软件系统的不同观点和方面。对于描述微处理器内软件结构，从环境或者接口观点，层观点和运行模型的观点是特别有利的。下面叙述在模型观点中用到的一些基本术语和符号。

4.2 基本概念

首先讨论一些基本的术语和概念，以及 V 模型[15]中的一些图形符号。

4.2.1 过程、处理步骤和产品

在开发过程模型的意义上，过程是指一系列系统性的和可重复的且具有一定逻辑顺序的步骤。一个过程可涉及：

（1）实现企业内部或外部客户的需求；

（2）由用户发起；

（3）以产品或服务的形式提供顾客，然后由顾客（客户/供应商的关系）支付。

一个过程步骤由一系列自成体系的系列活动构成，这些活动的结果是产品。细分过程的步骤不会产生有用的产品。产品是供其他处理步骤使用的一个中间结果。例如，电子汽车系统的产品是说明书或者软件组件。其他产品也可以是硬件、设定点发生器、传感器、或者执行器。

不同的过程步骤由接口连接，这些接口用来交换产品。在可视化的 V 模型[15]中，过程步骤和产品如图 4-3 所示。

在过程步骤和产品的定义中，必须考虑到多种限制：

（1）参加者或责任人，或者，谁负责什么？

（2）能力和资格，或者，谁能胜任某事？

（3）前提条件和结果，或者，需要什么和交付什么？

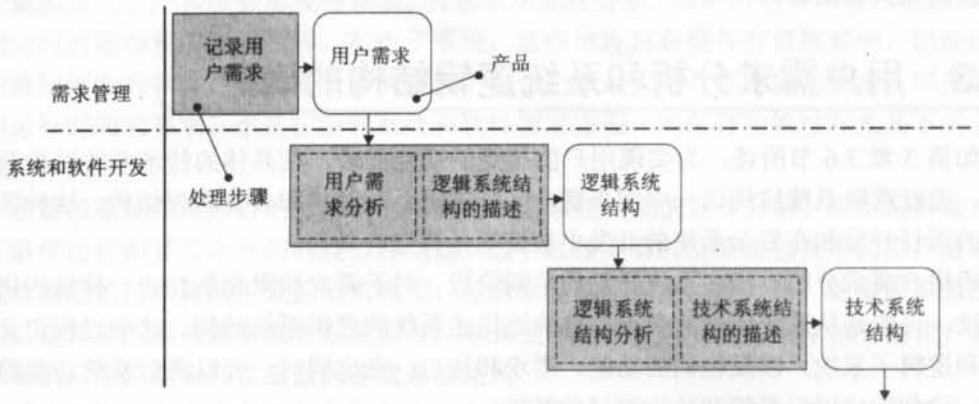


图 4-3　应用 LOV 图描述 V 模型中的过程[23]

4.2.2　方法和工具

对每一个过程步骤或一系列步骤，必须应用所谓的方法来处理。根据参考文献[72]的定义，方法是被证明是正确的、按预定的程序应用的、用于达到既定目标的步骤。

例子：一种新的汽车功能的仿真和快速原型

快速原型是指通过应用某种定义好的方法，尽可能早和快速地核实系统功能有效性的一种过程步骤。在软件功能的开发中，快速原型工具支持这些方法的应用（见第 1 章图 1-20）。在这方面，典型的处理步骤是系统功能的描述（例如通过建模）、仿真、集成以及在汽车中的测试（见图 4-4）。

这种方法产生的产品是系统功能的描述，它是以一种模型的形式，对于一种功能的要求尽可能完整且无矛盾的描述。例如，系统功能的描述是以一种可以执行的，且通过分析或实验的方式验证过的模型。这种系统方法对新功能的可行性分析是很有效的，除了减少开发的风险，通过软件和硬件的并行开发与测试，还可以尽可能地缩短产品开发的周期。

关于汽车的集成和测试，可以分为几个步骤，譬如采用实验性系统或实验室汽车，以及在道路行驶的真实汽车中实际性能的试验（见图 4-5）。任何没有为后续的过程模型提供可使用产品的开发步骤称为方法性步骤。

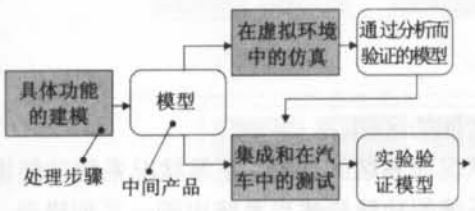


图 4-4　开发过程中的仿真和快速原型步骤

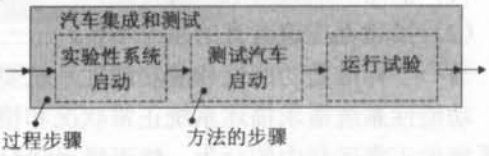


图 4-5　汽车中集成和测试阶段所采用的方法步骤

所谓工具，就是自动支持方法的应用。工具通过应用方法支持过程步骤的处理，因此有助于生产效率的提高。特别是对于需要高精确度或频繁重复执行或需要检验的那些过程

步骤来说是尤其有用的。

4.3 用户需求分析和系统逻辑结构的确定

正如第 3 章 3.6 节所述，为实现用户能接受的功能需求，在具体的技术系统结构开发过程中，通过逻辑系统结构这一中间步骤，区分逻辑系统结构和具体物理结构。这种区分在长期的项目开发和在复杂系统的开发中被证明是特别有效的。

术语用户需求分析，指在系统开发的早期阶段，对于需求和限制条件的一种结构化的处理方法。目的是从系统用户的观点，准确地描述系统的逻辑系统结构。这个过程定义逻辑组件和逻辑子系统，以及他们的功能、需求和接口。在这同时，可以确定系统功能的应用案例，这些构成以后系统和验收测试的基础。

如果需要，这个步骤也许一直循环直到逻辑系统结构满足所有的用户要求，并且直到系统和接受测试产生正确结果（见图 4-6）。

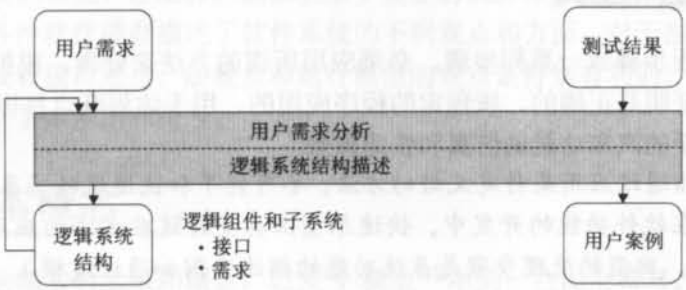


图 4-6 用户需求的分析和系统逻辑结构的描述

逻辑系统结构描述抽象结果，避免指定具体的技术系统结构。换句话说，系统将实现那些功能，但是没有具体的实现方法。逻辑系统结果被描述为系统和功能的抽象逻辑模型。这个模型构成用户要求和后续步骤的技术系统结构设计之间的连接。逻辑系统结构的定义是由接受用户要求开始的创造性管理和设计过程。与用户需求相比较，逻辑系统的结构要求是用参与开发过程的工程学科的语言来表达。图形化标志，譬如结构框图和状态自动机，适合于基于模型的逻辑系统的描述。

逻辑系统要求可从两种不同的观点来描述：

（1）描述应该具有的系统特性；

（2）描述不应该具有的系统特性。

另一个标准是功能性和非功能性系统要求之间的区别。

功能性系统需求描述系统正常状况和错误状况时系统的功能。正常状况系统功能描述在系统的正常运行中的行为，然而错误状况时系统的功能描述当系统出现一系列错误、故障和失误时的系统运行。

非功能要求描述系统的所有附加要求，这些也被称为限制。他们形成的类别包括，例如变量和可量测性要求，通过立法而规定的限制，例如可靠性和安全要求。同样的种类包

括大量的通过生产和服务形成的要求。其他非功能性要求，如车内电子网络的电压可靠性，装配空间的限制和成本限制等。在电子系统，这些也许只对硬件有直接影响，但没注意到这些将间接影响软件。硬件需求影响软件的例子，如最大可允许的资源需求，即存储器数量和运行时间的要求。正是在逻辑系统和软件要求方面，汽车开发的过程尤其不同于其他领域。

在逻辑系统结构分析的过程中应用广泛的是系统功能的逐步分解，应用这种观点不仅决定系统组件而且决定他们的接口和功能。这个过程步骤的结果是包含不同用户要求和所有系统功能的结构化的正式的结构模型。逻辑系统结构也指功能网络。如果逻辑系统结构指的只是特殊的系统版本或者系统型号，那么逻辑结构可能只是包含了部分的用户需求。

举例：用户需求和仪表盘的逻辑系统结构

图 4-7 显示了在汽车仪表盘完成的一些用户需求。这个仪表盘例子在这章将作为核心开发过程每个单一开发步骤的应用实例来讨论。

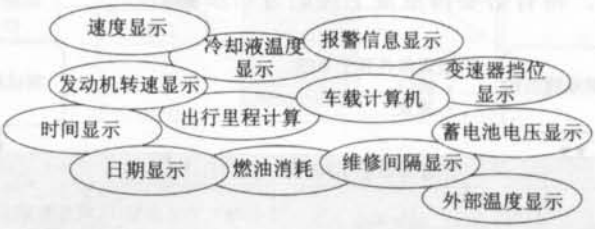


图 4-7　对于汽车仪表盘的部分用户需求

图 4-8 显示了在系统逻辑结构上实现的用户需求。逻辑结构图采用分层次的结构是基于设计标准的考虑。例如，可视化类型将提供可选择的数字显示或指针仪表。因此，在逻辑系统结构的开发中，为了用显屏或指针仪表显示信息的处理过程，已经进行了装置预设（如通过滤波器或衰减功能）。

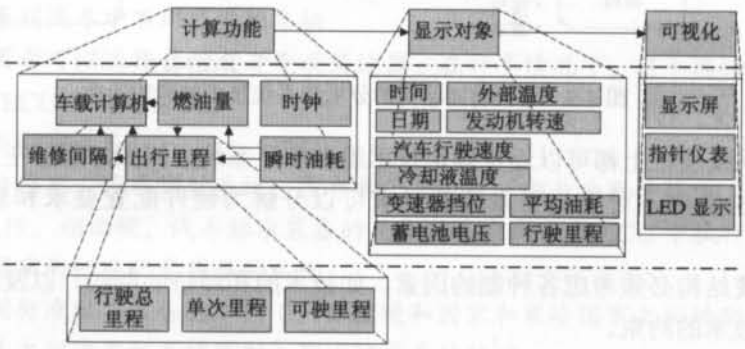


图 4-8　汽车仪表盘的系统逻辑结构

如果用户要求不能与逻辑系统结构之间进行区分，随着项目开发步骤的逐步进行，我们将难以识别出用户原先真正所期望的系统的要求。而且，将完全不能预料用户的要求是否真正在系统中被实现。

这步的开发目标是获得逻辑系统结构的定义，得到确定的、不冲突和完整的逻辑系统结构。处理的步骤是把非正式的、不完全的、无结构的以及口语化的用户要求的描述转化为结构化的功能模型。

在系统的逻辑结构方面的几个结构公理，例如易管理性、可分离性和可理解性，提供重要的实质性的引导。由此形成逻辑模型、定义系统的功能和功能之间的接口。

4.4 逻辑功能结构分析和系统技术结构的确定

在逻辑系统结构的基础上，系统技术结构的描述将确定具体的实现方面的决定。把逻辑系统的要求被分配到技术组件和子系统后，执行一些基本的分析，诸如开环和闭环控制任务分析，或者采取分布式的和网络系统的分析，或者可靠性和安全性分析（见图4-9）。主要目标是评价各种技术实现方式，所有这一切都是基于一个标准的逻辑系统结构。关于实现方式决定的变化，将有必要再重复上述的分析步骤。

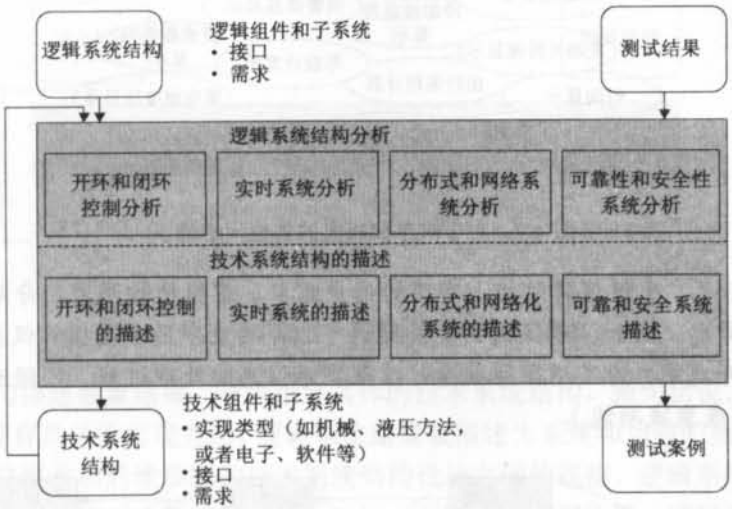


图4-9 逻辑功能结构的分析和系统技术结构的确定

在所有系统层次上都可以逐步确定技术组件和子系统。例如，随着在上一级系统层面的实现方式的确定，对于电子系统的要求可以分解为硬件配置要求和软件需求（见图4-10）。

技术系统结构必须考虑各种制约因素，如技术的和经济的制约，以及那些相关的组织结构和制造技术的约束。

下面是一些典型的限制：

（1）标准和设计方式；

（2）各系统和组件之间的相互依赖；

（3）可行性分析研究的结果；

（4）生产和服务要求；

（5）可修改性和可测试要求；

（6）成本和风险评估。

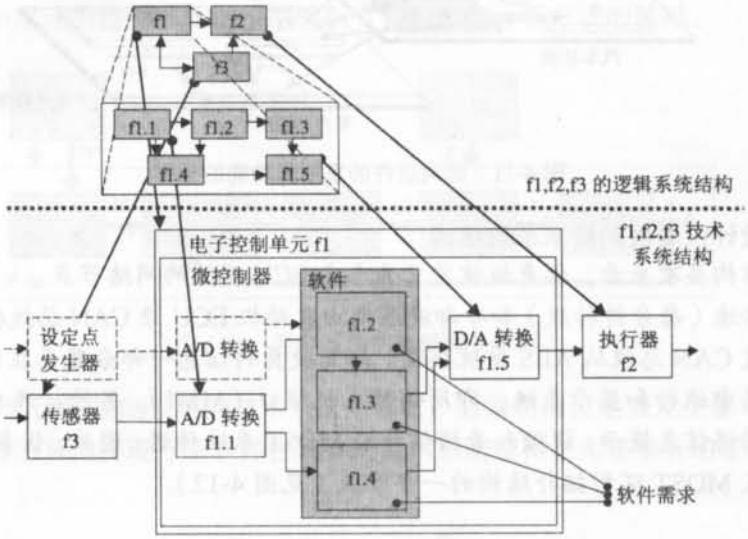


图 4-10　系统技术结构的确定

多个相关的工程学科领域专家的参与是确定系统技术结构的前提条件。更多的时候，这个过程还需要解决目标的冲突。

举例：在确定系统的技术系统结构时，限制条件和目标的冲突

（1）在不同系列汽车中，技术组件的重用

在多个不同的汽车型号系列中，重用发动机和驱动系统是出于成本的考虑。因为这个事实，也影响了电子系统的结构，这经常是导致一体化的发动机和驱动系统，只根据程序和数据版本区分不同的发动机和驱动系统的原因。

（2）同一系列汽车中不同类型的车辆

汽车的购买者可以选择自动或者手动变速器。在许多情况下，这个选择引起在发动机 ECU 和变速器 ECU 之间的分离。

（3）附件与标准设备

因为雨点传感器、停车辅助系统，或者电子座位校正都作为附件提供，它们需要依靠另外的 ECU 执行。相比较，汽车标准装备的几个功能在同一个 ECU 中执行。

（4）不同国家间标准配置装备的变化

不同国家间标准配置装备的不同（例如在暖和国家和寒冷国家之间的配置的不同，或者在欧洲和北美之间使用版本的区别）影响技术系统结构。

（5）针对组件的重用

重用的目标是方便跨品牌和厂商应用组件。如果是这样的话，面向组件的重用优先于任何功能的分解。图 4-11 描述了强制重用 ECU1。它包含功能 f1、f2 和 f3。然而功能 f4 可以分配给其他 ECU，在这个例子中，如可分配给 ECU3。

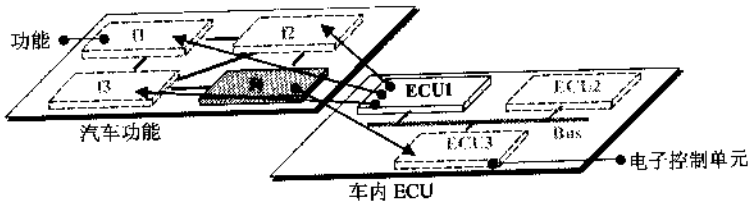


图 4-11 面向组件的重用和功能的分解

举例：设计仪表板的技术系统结构

如果从结构层次来看，仪表板被定义为车内 ECU 网络的网络节点。

发动机转速（每分钟转数）和冷却液温度由发动机 ECU 经 CAN 总线提供。汽车的路面速度也经过 CAN 总线从 ABS 系统接收。给驾驶员的信息一部分显示在仪表板，而另一部分显示在集中运行和显示系统，即所谓的人机接口（MMI）。另外，汽车音响系统可以产生警告和错误信息提示，视频和音频信号经 MOST 系统传送。因此，仪表板设计成 CAN 总线站点以及 MOST 环型拓扑结构的一个节点（见图 4-12）。

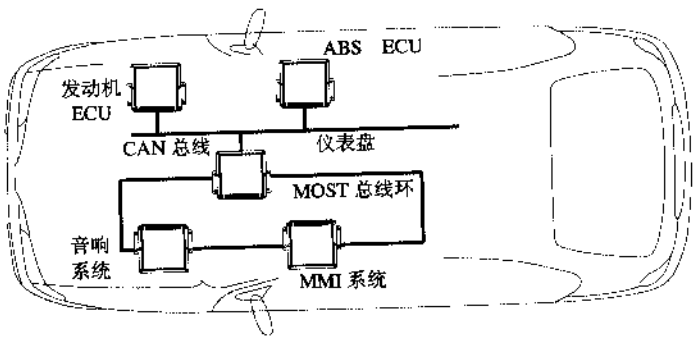


图 4-12 汽车控制器网络的技术系统结构

燃油液位指示器（作为传感器）信号也显示在仪表板上，（作为执行器）显示在硬件系统结构上，如图 4-13 所示。

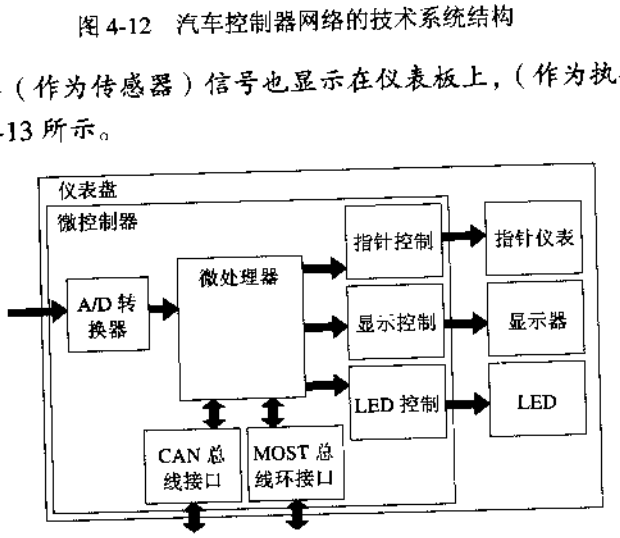


图 4-13 仪表盘硬件的技术系统结构

4.4.1　开环和闭环控制系统的分析和技术结构的确定

如图 4-14 中描述的逻辑系统结构形成所有控制系统分析方法的基础。

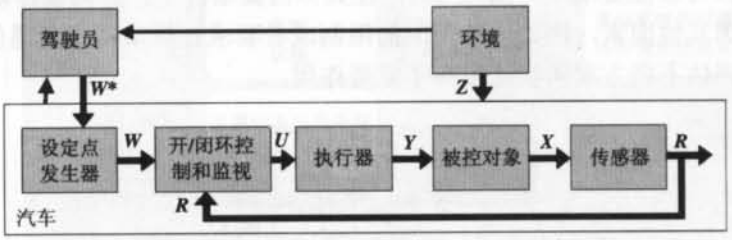


图 4-14　控制系统的逻辑体系结构

当确定开环和闭环控制系统的技术系统结构时，必须明确设定点发生器、传感器、执行器，ECU 网络的具体实现方法。并在具体的技术系统结构上实现系统的逻辑体系功能，如图 4-15 所示。

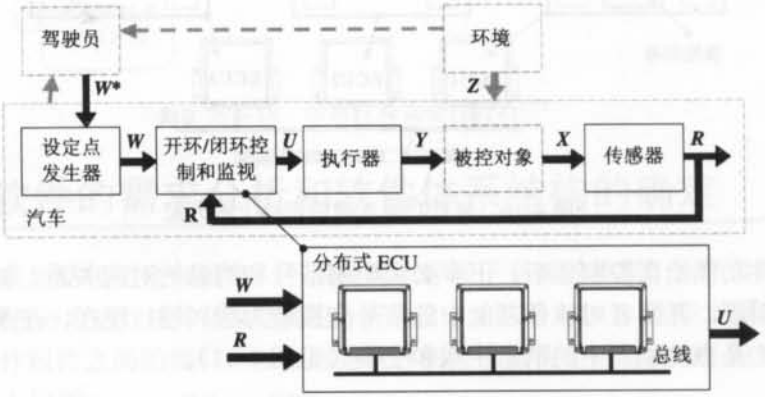


图 4-15　控制系统的技术体系结构

4.4.2　实时系统的分析和确定

在控制系统的分析与确定技术系统结构时，也必须确定对于不同控制性能的时间采样速率要求。

采样率是确定软件功能实时要求的基础，软件功能将在各个车载 ECU 的微型控制器中执行。在实现包括一个分布式和网络的系统情况下，根据采样速率，也必须对通过 ECU 间的通信系统进行传输的数据传输的实时性提出要求。

在给定的实时性要求下，依据微控制器中的实时操作系统，对于系统实时要求的可调度性分析方法见第 5 章 5.2.2 节。这个分析结果作为评估技术实现方法和修改实时操作系统配置的依据。因而原则上，这个方法也可扩展到分析通信系统的实时性。

4.4.3 分布式网络控制系统的分析和确定

分配逻辑软件功能到微控制器网络系统中，这一开发步骤需要考虑多种因素，例如实时性，安全性和可靠性要求（见图 4-16）。在具体前提条件下，分配软件功能到不同的微控制器需要考虑其他因素，例如安装空间的限制或者要求计算功率或者通信能力。因此，在分析阶段，评估不同方案的实施发挥了重要作用。

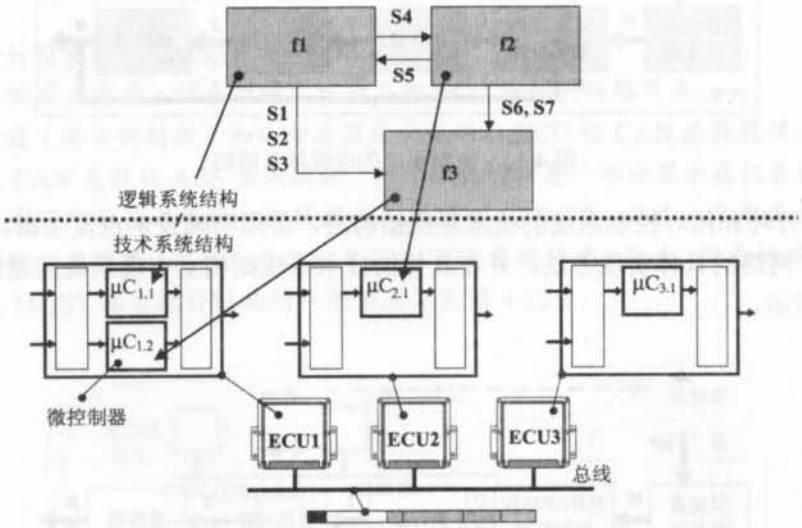


图 4-16 软件功能在微控制器上的分配

分配软件功能给微控制器后，下步必须明确信号和消息的对应关系。如前所述，在系统结构的逻辑层，开发者对软件功能中的信号交换最为感兴趣。现在，在系统结构的技术层，感兴趣的是微控制器中的消息生成和交换（见图 4-17）。

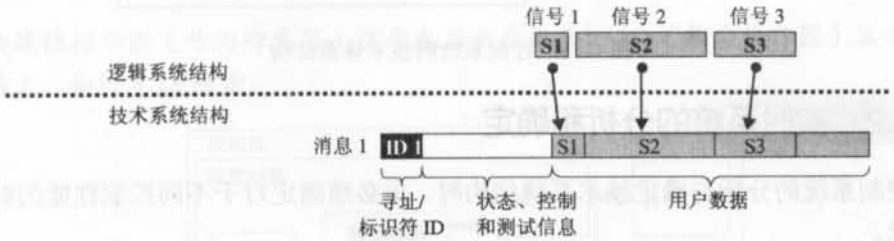


图 4-17 信号在消息上的分配

4.4.4 可靠性及与安全相关系统的分析和规范

对许多汽车功能来说，可靠性和安全性的核查是一个强制要求。为此，可靠性和安全性的分析必须在早期的开发过程中确定。图 4-18 给出了可靠性和安全性分析的典型过程[70]。

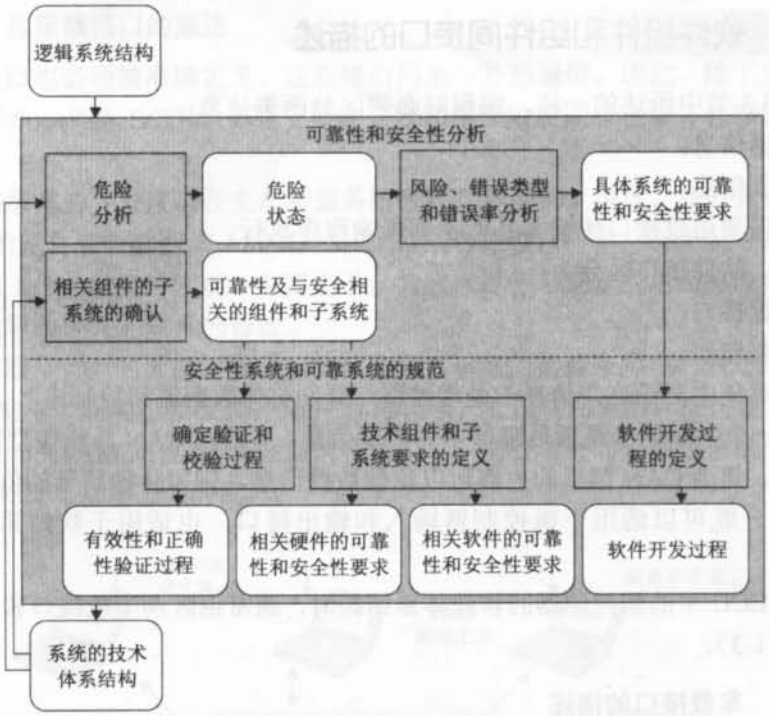


图 4-18　可靠性和安全性分析[70]

4.5　软件的需求分析和软件体系结构的确定

随着技术系统结构的全部确定，下步处理组件和子系统的实现。从软件需求分析开始（见图 4-1），软件开发进行软件体系结构的分析和设计（见图 4-19）。这些步骤包括确定软件系统界限、软件组件之间的接口，以及确定软件的层次结构与运行状态。这些步骤应用于软件开发的各个层面。

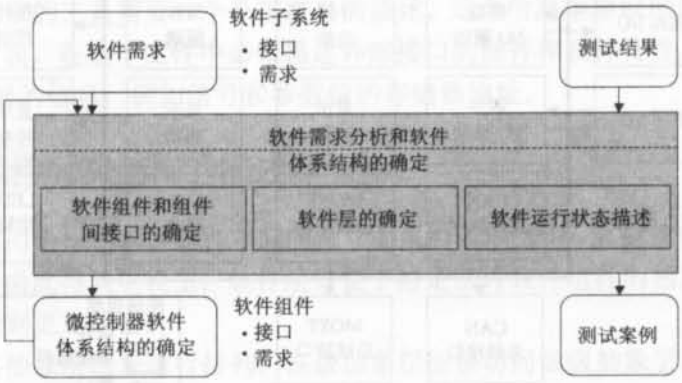


图 4-19　软件需求分析和软件体系结构的确定

4.5.1 软件组件和组件间接口的描述

如在 2.3.3 节中所述的一样，编程时必须区分两类信息：

（1）数据信息；

（2）控制信息。

两者都需要通过接口传输并且两者都影响程序执行。

相应地，软件接口区分为：

（1）数据接口；

（2）控制接口。

在软件系统中对于这些信息的处理过程，分别称为数据流和控制流。

例如，一个中断信号是微处理器中的控制信息，即通过 CAN 总线模型接受 CAN 信息来触发中断。然而 CAN 信息的内容可以称做数据信息，例如传输信号的值。

这些区分既可以适用于微控制器输入和输出接口，也适用于软件系统内部组件间的接口。

在设计 ECU 中的微控制器的软件体系结构时，通常也区别车载接口和非车载（车外）接口（见图 1-3）。

4.5.1.1 车载接口的描述

第一步，软件系统的界限必须被准确地定义。这个任务需要项目所有参与者合作协商。目的是决定哪些属于软件系统的一部分，哪些属于外围或者环境。只有这样才能确定软件的输入和输出接口。同样的方法可以用来确定车载 ECU 的接口，即 ECU 与设定点发生器、传感器和执行器的接口，以及与车内其他电子系统内部通信的接口。

举例：汽车仪表盘的软件的接口模型

图 4-20 描述了汽车仪表盘软件车载接口的接口模型。

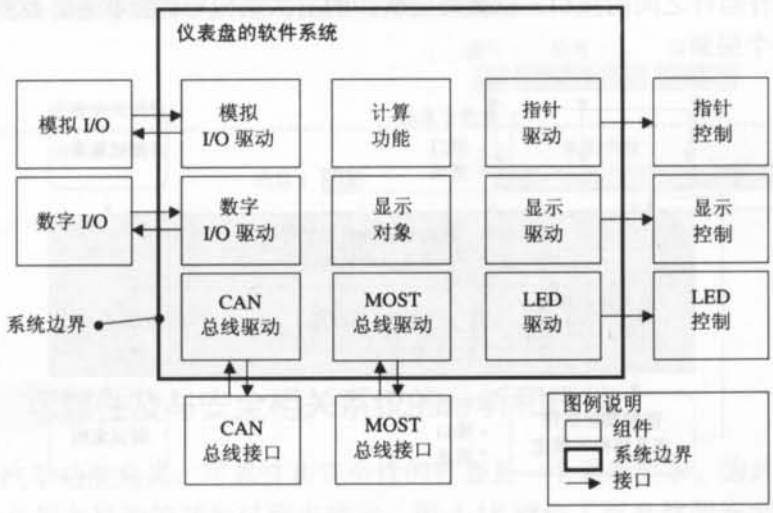


图 4-20 汽车仪表盘软件的接口模型

4.5.1.2　非车载接口的描述

另一组接口也必须被准确定义，这些接口用来与外部通信。因此，除了完全支持车载的功能与接口外，产品 ECU 也必须支持在开发和生产阶段或者在服务修理阶段跟外部通信的接口。

因为不是所有的开发接口在生产和服务阶段都是需要的。因此，在开发过程中，应用不同的 ECU 的硬件和软件版本，例如开发 ECU 或者原型，样本，或者应用 ECU。这些控制单元与产品 ECU 之间的区别，主要在于：为适应每一个开发应用而修改相应的非车载接口，并由此导致硬件和软件的修改。

工具和车载 ECU 的微控制器之间的通信通过不同的接口来实现。对于测量、校准、诊断和 Flash 编程等功能已经在 ASAM 中标准化了。对这些功能的详细讨论，参见相关的专业文献（例如 ASAM 规范）。当设计软件结构时，需要考虑与外部通信的软件模块。图 4-21 显示了 ECU 与常用工具的接口。

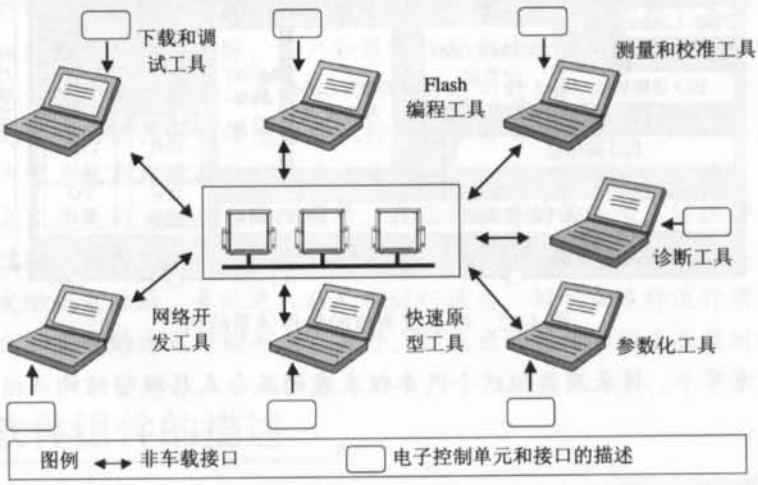


图 4-21　控制器的几种可能的非车载接口

每一个列出来的工具需要一个外部接口的描述。这个信息主要以所谓的描述文件的形式来提供。一方面，在这个文件中必须描述外部接口的硬件和软件特性，另一方面，它需要存取 ECU 数据的信息，例如信号和参数值的存储器地址。

4.5.2　软件层次结构的描述

经常用到的用于描述软件组件之间的相互关系的组织结构，是把软件组件与不同的软件层联系起来，由此得到层模型。软件层方便了给定层内软件组件的相互存取。然而，不同层间存取的规则更加严格。

层结构按其抽象层次来进行排列。高级抽象层能够访问低级抽象层。相比较而言，而从低级抽象层访问高级抽象层通常或有严格的限制，或者是不允许的。软件结构允许从高抽象层访问低抽象所有层，这叫做严格顺序层模型。然而，如果在所有情况下，访问都被

限制在高抽象层向低其一级的层次访问，这个叫做线性顺序层模型。

例如，根据 OSI/ISO（见图 2-50）的七层 OSI 参考模型是线性顺序层模型，OSEK-COM 通信模型以线性顺序模型作为其结构。层结构也经常引入其他 I/O 接口，如平台软件中的接口和应用软件内部的接口。抽象层次的引进简化了软件组件的设计，有助于软件组件的维护以及重用。

举例：仪表盘的软件结构

考虑到标准和设计方式，仪表板的软件结构设计如图 4-22 所示。

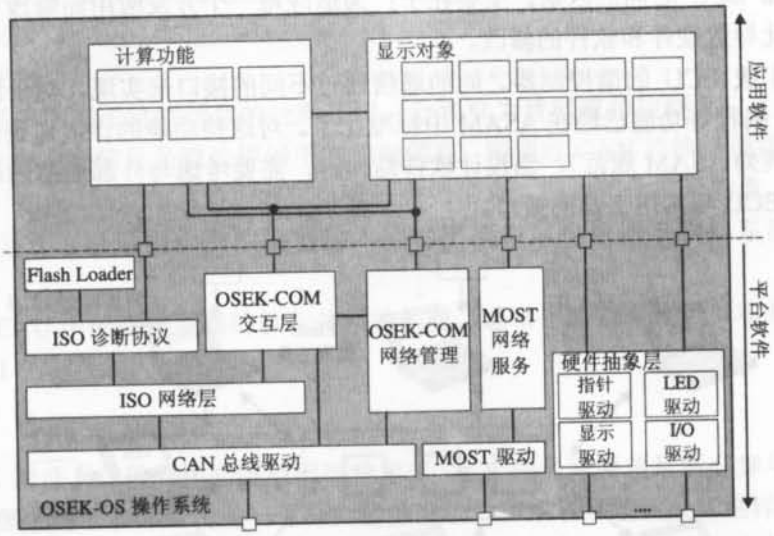


图 4-22 汽车仪表盘的软件体系结构

在下面的章节中，将采用类似这个汽车仪表盘的混合式层模型结构来描述软件系统的体系结构。

4.5.3 软件运行状态的描述

在生产和服务阶段中，软件变量的参数化和软件更新大多处于软件的特殊运行状态，在这些运行状态，出于安全的考虑，控制和监视功能允许部分处于运行状态或完全停止运行状态。

因此，车内 ECU 的微控制器的软件结构必须被设计成支持不同的软件运行状态。除了在正常的运行状态外（在该状态，执行控制和监视功能，所谓的行驶程序），通常还需支持其他几种运行状态，在这些运行状态，行驶程序停止执行。例如，执行器的诊断状态（见图 2-64），软件参数化和更新状态。

同样，紧急运行模式（见第 2 章 2.6.4.4 节），通常在与安全相关的组件失效后，系统以有限功能运行的状态，也可以看作一种独立的运行状态。

除了界定运行状态外，还必须明确那些状态之间允许相互转换，以及转换的条件。因而，状态机是描述系统状态以及状态之间切换的合适的工具。

举例：驱动仪表盘软件的运行状态

如图 4-23 所示，仪表盘软件需要如下的运行状态及其相互间转换：

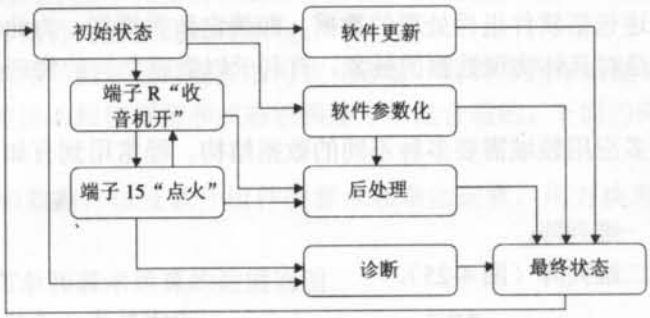


图 4-23　汽车仪表盘工作状态及其转换

（1）整个显示功能在端子 15 发动机点火的运行状态提供。

（2）在运行状态为软件更新时，在生产和服务中通过外部诊断接口支持 Flash 编程。

（3）运行状态为软件参数化时，生产和服务经外部诊断接口设置软件参数。例如将距离和速度显示从公里转换到英里，或者在不同语言之间的切换。

（4）固有的诊断功能，例如传感器和执行器的诊断功能（见图 2-64），以及读出或擦除错误的存储内容，只有在诊断运行状态才有效。

（5）转动点火开关到 Radio ON 位置时，在运行状态为端子 R，将执行大量的监视功能。只有所有这些功能都已运行后，切换到端子 15 的运行状态才会生效。

（6）关掉发动机电源后，系统进入后处理运行状态。例如在这种运行状态，汽车的整个里程数以永久和安全的方式存储起来。另外，仪表盘的监视功能也在这时执行。

4.6　软件组件的描述

在确定软件体系结构的阶段，所有的软件组件、要求和他们间的接口已被定义。现在集中讨论软件组件的描述。这里将区别软件组件的数据模型、动态模型以及软件组件的实时模型（见图 4-24）。

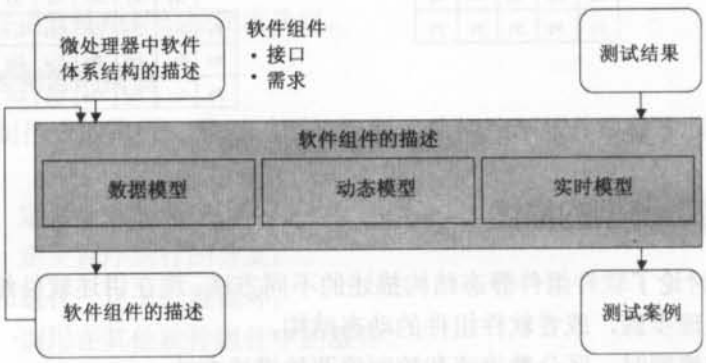


图 4-24　软件组件的描述

4.6.1 数据模型的描述

软件组件的描述包括软件组件处理的数据，即确定数据模型。为此，首先确定一个抽象的数据形式，它是对具体实现数据的抽象，有利于以数据之间的物理关系来定义对数据的处理。

在汽车中的许多应用领域需要多种不同的数据结构。经常用到有如下结构：

（1）标量；

（2）矢量，或一维列阵；

（3）矩阵，或二维列阵（图 4-25）。

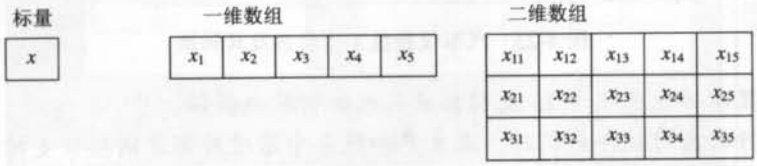
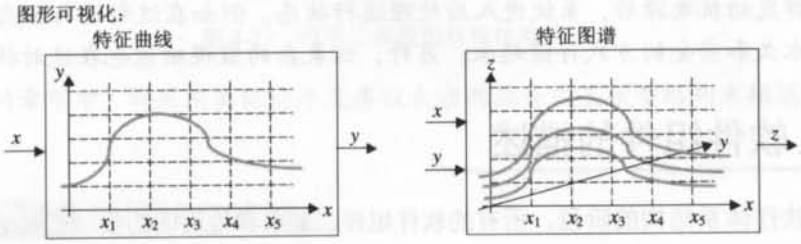


图 4-25 简单的数据结构

数据之间可能存在相互关联，例如构成组合的数据结构。例如，被广泛应用的特征曲线和特征图的数据结构（如图 4-26 所示）。在抽象的特征曲线或特征图描述的情况下，我们只关心输入和输出变量之间的相互关系；然而在设计和实现时要求确定表格的存储方式及插值算法。不同的图表存储方法和插值法将在第 5 章 5.4.1.5 节详细讨论。



表格形式：

x_1	x_2	x_3	x_4	x_5
y_1	y_2	y_3	y_4	y_5

	x_1	x_2	x_3	x_4	x_5
y_1	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}
y_2	z_{21}	z_{22}	z_{23}	z_{24}	z_{25}
y_3	z_{31}	z_{32}	z_{33}	z_{34}	z_{35}

图 4-26 复杂的数据结构

4.6.2 动态模型的描述

目前为止，讨论了软件组件静态结构描述的不同方法。现在讲述软件组件的动态行为，即软件组件的处理步骤，或者软件组件的动态结构。

在描述动态模型时，区分数据流和控制流两种描述方法。

4.6.2.1　数据流的描述

数据流图提供两个基本类型的信息。一方面，描述软件组件间数据信息流的路径。另一方面，显示软件组件内数据的处理流程。

有多种不同的用于描述数据流图的可视化方法。对许多汽车功能来说，在本书 2.1 节介绍的控制系统建模的结构框图和状态机模型方法是合适的。下面的例子也是用框图的形式来描述数据流。

输入、输出和数据，以及软件组件的算术和布尔运算，用方块表示。数据流用箭头表示。

举例：用于布尔和算术运算的数据流图

图 4-27 中分别表示布尔指令

$$Y=X1\&(X2 \parallel X3)$$

和算术指令

$$c=a+b$$

其中，&表示逻辑与运算，‖表示逻辑或运算。

在 ASCET 工具中以结构框图形式描述数据流。

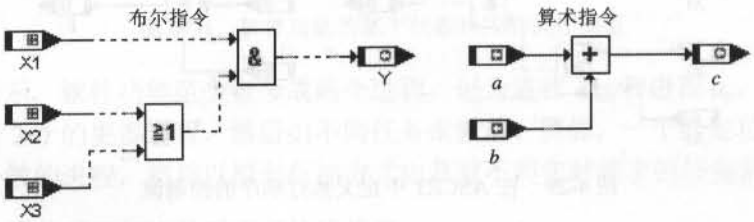


图 4-27　在 ASCET 中布尔和算术关系式的表述

算术数据流用带实线的箭头来画；布尔数据流用带虚线的箭头来画。

虽然数据流图易画并且易懂，但不能对软件组件的动态特性提供完整的信息。

例如，图 4-27 没有指明是布尔指令先执行还是算术指令先执行。在这个简单例子中，应当指出，由软件组件产生的结果不随执行顺序改变而改变。然而，如果结果 Y 和 c 彼此相关，那么 c 的值将会进入 Y 的计算结果，因而执行顺序的变化将改变软件组件的动态特性。执行的顺序由软件组件的控制流决定。

4.6.2.2　控制流的描述

控制流控制指令的执行。通过以下的控制流结构控制软件组件中指令执行的过程和次序：

（1）**顺序**　定义处理的顺序；

（2）**分支**　定义程序运行的分支；

（3）**重复和迭代**　定义处理循环；

（4）**调用**　调用在其他软件组件中的服务。

这种控制流结构在任何高级程序语言中都可以找到，并可以用图形化的方式来表达。

其中的一种知名的控制流结构的记法是根据 Nassi-Shneiderman 的结构程序，也叫 Nassi-Sheiderman 图（见图 4-28）[72]。

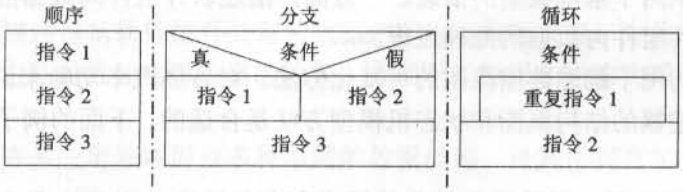


图 4-28 控制流结构的图形化描述

然而，在许多软件工程中，仅仅有控制流或仅有数据流是不够的。通过一种组合的描述方式是必要的。

举例：布尔运算和算术运算指令的控制流

图 4-29 显示了在前一个例子中没有的序列信息已经被加入到结构图中。控制流结构决定了算术指令 $c=a+b$ 被分派进程 1 的次序信息/1/，并且比布尔指令 Y=X1&(X2 || X3)（被分配进程 1 的次序信息/2/）先执行。

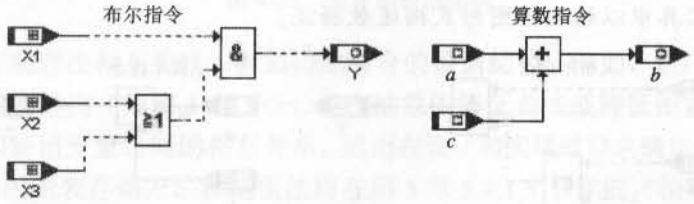


图 4-29 在 ASCET 中定义执行顺序的控制流

4.6.3 实时模型的描述

为了软件组件描述的完整性，除了前面部分讨论的数据和动态模型，还必须确定实时模型，这就意味着软件组件的指令必须分配给具体的进程来运行，而进程又必须分配给某个任务，对于任务定义实时性的要求。

例子：实时性要求的定义

在图 4-29 中介绍的例子，两个指令都已分配给进程 1。为了完成实时性要求的描述，进程 1 必须分配给一个任务，这个任务见图 4-30。进程 1、进程 2 和进程 3 指派给任务 A。

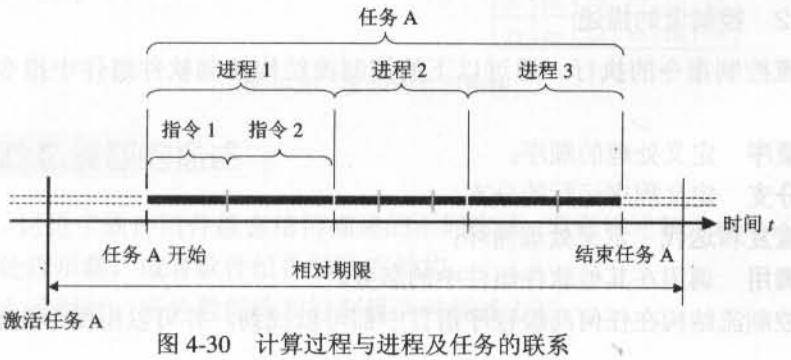


图 4-30 计算过程与进程及任务的联系

4.6.3.1　依赖于状态的反应式执行模型

对于执行控制任务的软件功能，通常建立一个通用的执行模型，如图 4-31 所示。软件功能区分为：初始化计算（在系统启动后执行一次）和循环计算（重复或者迭代执行）。初始化部分即进程 a_{Init}，由初始化任务在时刻 t_{Init} 来激活。循环计算部分即进程 a_{P}，由任务 A 每间隔时间 $\mathrm{d}T_{\mathrm{A}}$ 周期性地反复来激活，这个间隔可以是常量也可以是变量。进程 a_{P} 第一次执行所用的状态信息由初始化进程 a_{Ini} 来提供，接下来进程 a_{P} 的反复执行利用先前执行的进程 a_{P} 的状态信息，这个信息可以是先前计算的结果，或者是距离前步执行的时间间隔长度 $\mathrm{d}T_{\mathrm{A}}$ 迭代，具有这些特征的软件叫做依赖于状态的反应式系统[74]。

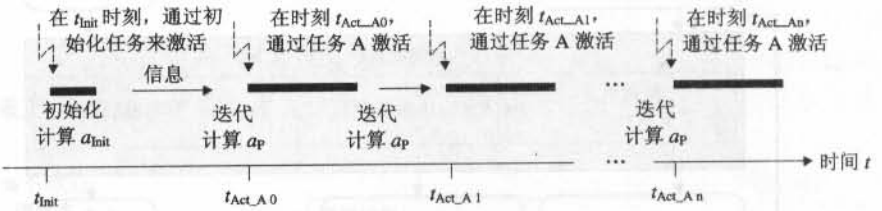


图 4-31　软件功能的基于状态响应的执行模型

如讨论所示，软件功能至少被分成两个进程，记为进程 a_{Init} 和进程 a_{p}。通常软件功能被划分成多于 2 个的更多进程，然后由不同任务来激活。例如，一个给定功能的循环计算被分为几个分散的进程，然后以拟平行的方式由具有不同实时要求的任务来激活。

4.6.3.2　不依赖于状态的反应式执行模型

在某些情况下，假定初始化的进程执行后，在重复执行部分，只有某个特定的事件出现时进程才能执行，而与这之前的进程执行的结果无关（前个进程的执行对于下个无影响）。在这种情况下，应用另一不同的执行模型，如图 4-32 所示。在这个例子中，软件功能分为两个进程，记为进程 b_{Init} 和进程 b_{E}。然而，进程 b_{E} 只能在任务 B 中的事件 E 出现时才能激活。这个事件 E 可以是，例如驾驶员按下某个控制软件功能执行的按钮开关。

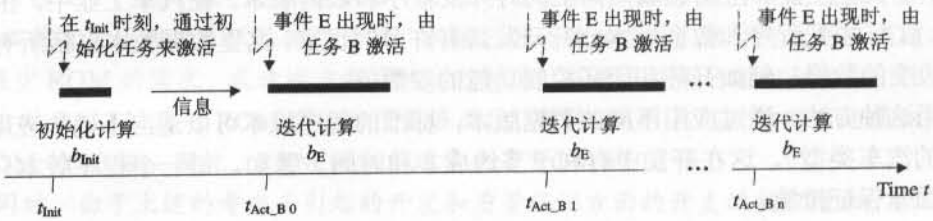


图 4-32　软件功能的非基于状态响应的执行模型

同样进程 b_{E} 能可以重复执行。然而与前面的例子相比，进程 b_{E} 的执行不需用任何先前执行进程 b_{E} 的状态信息，具有这些特征的软件功能也叫做不依赖于状态的反应式系统。

通常，混合使用两种执行模型。

在实时系统中，当描述不同任务进程间的相互作用时，必须考虑到在本书 2.4.6 节中描述的实时操作系统的机制。因而，实时模型经常影响数据模型。

4.7 软件组件的设计和实现

在软件组件的设计阶段，必须确定软件组件的数据、动态和实时模型的具体实现细节（见图 4-33）。对于数据，现在必须进一步区分变量和固定的参数值。

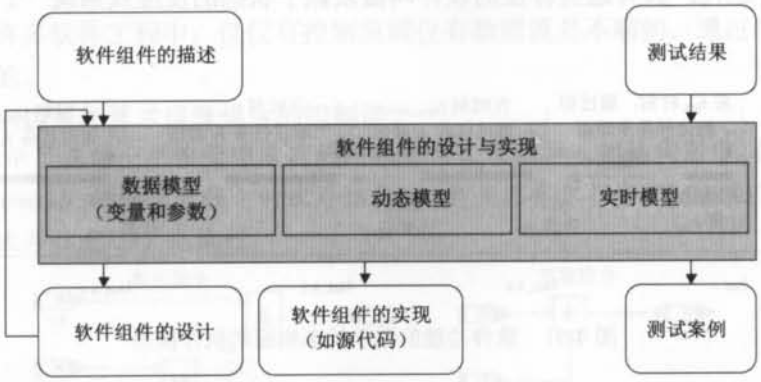


图 4-33 软件组件的设计与实现

4.7.1 考虑非功能性的产品特性要求

当为产品 ECU 设计和实现软件组件时，除指定的软件功能之外，必须考虑到非功能性的产品特性的要求。例如，这些非功能性的特性要求包括分离的程序版本和数据版本，也可能是经费限制而导致可靠硬件资源的限制。

4.7.1.1 程序版本和数据版本的划分

程序版本和数据版本的划分，不仅在开发期间而且在生产和服务期间，均有利于简化对软件变量的处理。

然而其他工业和应用领域倾向于同时开发程序和数据版本。在汽车工业中，由于许多原因，ECU 中把程序和数据版本分开开发具有许多的优点。这里数据版本包括所有不被程序所改变的数据，例如开环和闭环控制功能的参数值。

用这种方法，通过应用不同的数据版本，标准的程序版本可以适应不同的应用（例如不同的汽车类型）。这在开发中有利于节约成本和时间。譬如，每一个程序版本只需应用一次质量保证措施。

采用程序和数据版本分开方法的其他原因还有：

（1）程序版本和数据版本的产生时间有时是不同的。例如，在具体汽车上运行软件校准功能的过程中，经常需要修改数据版本，数据版本也经常在一个很晚的开发阶段独立于程序版本而改变。

（2）创建程序版本和数据版本经常发生在不同的开发环境中并且由不同的团队成员操作，甚至是跨单位合作。

（3）独立的程序版本和数据版本不仅在开发中好用，而且在生产和服务中关于变量管理方面获得了实际的利益。

4.7.1.2　硬件资源的限制

设计和实现软件组件时，由于可以利用的硬件资源的限制，经常需要考虑优化措施。可以看出，根源之一是成本的限制，并由此导致产品的大批量生产时对于产品硬件资源的限制。

举例：ECU 的成本限制

计算 ECU 成本的简单方式是，ECU 的成本是开发成本和制造成本的总和除以生产单元总数 n：

$$每一个\ ECU\ 的总成本 \approx （开发成本+生产成本）/n$$

ECU 的最终成本，主要由制造成本所占的比例来决定（见图 4-34）。

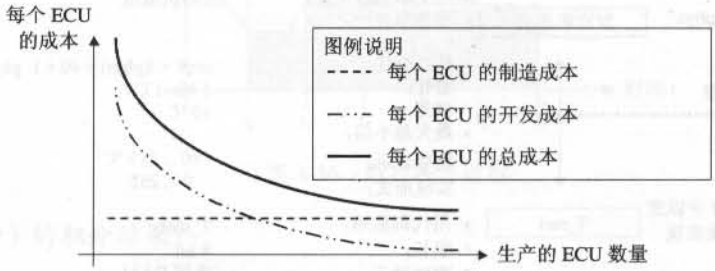


图 4-34　单个控制器的成本与生产的产品数量之间的关系

在这种情况下，软件和硬件的成本结构是不同的。如果假定软件复制的成本接近零，那么生产成本主要由硬件成本影响，成本与生产单元的数量成比例。

在 ECU 的生产数量很大的情况下，降低硬件生产的成本的压力是很大的。因而，在产品 ECU 中经常应用价格比较便宜的微控制器，由此造成了硬件资源方面的限制。因而，这些微控制器通常只支持整数算术运算，并在计算能力和存储量方面是比较有限的。

在这种情况下，为了在一台微控制器中处理尽可能多的功能，软件开发过程中应用优化措施，以最有效地利用有限的硬件资源。根据应用的限制条件，软件开发的目的是减少 RAM 或减少 ROM 的需求，或者减少程序的运行时间。在许多应用的微控制器中，作为基本的法则，对 RAM 应比对 ROM 需要给予更多关注，因为 RAM 在芯片中所占的面积空间大约是 ROM 的 10 倍。因此，RAM 的成本通常也是 ROM 的 10 倍。

与此同时，由于上述的考虑而引起的开发和质量保证方面的开支的相应增加，以及在资源有限时，在开发中可能增加的质量风险，这些都不能被忽略。

因而，在实际应用中，制造成本的优化结合在硬件和软件组件方面采用的标准平台的方法被证明是有优势的。例如，采用不同的配置参数，标准软件组件可以适合一个特定应用的要求。

不同的项目目标（如质量目标、成本目标和时间表或者期限限制）之间通常存在着相互依赖和冲突情况。为此，必须注意到，不应由于优化措施而带来的附加的过多的花费和代价。

实际中应用的许多优化措施将在第 5 章 5.4.1 节的例子中讨论。许多优化措施也可能对软件结构和软件组件的描述带来影响和改变。

4.7.2 数据模型的设计和实现

当设计和实现软件组件的数据模型时，要区分变量和参数值两种不同的数据类型，参数值不随程序而改变。对每一个软件组件，设计阶段决定所有数据在处理器的内部表示法，以及考虑在微控制器存储器中的数据存储方式。因而变量必须被存储在读/写存储器，如 RAM，然而参数存储在只读存储器中，如 ROM。

举例：功能要求的描述与具体实现之间的转换

分别在功能要求的描述和具体实现的层次上，图 4-35 显示发动机温度信号的表示法。

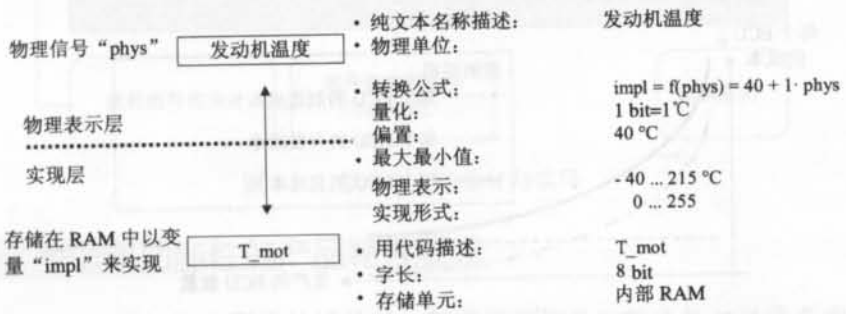


图 4-35 功能要求描述的具体实现

诸如在测量和校准情况下，必须从具体实现转变到物理表示法。原因是在测量和校准工具中，具体实现的变量应该被转化为具体的物理单元来显示。为了满足这个显示需求，测量和校准工具需要所有相关的数据信息。这个信息被存储在说明文件中。ASAM-MCD2 标准中规定了测量、校准和诊断功能组件的描述格式。

4.7.3 动态模型的设计和实现

除了动态模型的描述，动态模型的设计还必须考虑处理器内部的数据表示法和数据处理对于算术运算结果精度的影响。数字处理器计算结果的精度由不同的类型误差所限制。主要有以下几种误差类型：

（1）输入数据的误差；

（2）舍入误差；

（3）近似误差。

如果输入数据由有限的精度或分辨率的测量变量来提供，输入数据的误差很难避免。由应用在汽车内的传感器提供的数据和处理器得到的结果总是会出现这种误差。这类误差

也称为量化误差（见第 2 章 2.2.2 节）。

舍入误差发生在用有限位数字计算处理的情况，事实上是微控制器配置的问题。例如，在定点运算中舍入误差是不可避免的（例如应用有限位数字做除法或者测量的结果）。

近似误差依赖解决问题所采取的计算方法。即使应用的计算方法没有舍入误差，许多计算方法也不能产生我们想要的结果。就是说，他们只得到简单问题 P*的结果来代替问题 P 的结果，是实际问题 P 的一个近似解。通常我们通过离散化原始问题 P 得到近似问题 P*。例如，微分是通过差分来近似，积分是通过有限的总和来近似。近似误差在定点和浮点运算中是不可避免的。

举例：欧拉积分方法

欧拉积分方法的例子如图 4-36 所示。此方法经常用在 ECU 中（例如，图 2-2 描述的 PI 控制器执行的积分运算）。这里，函数 $f(t)$的积分是通过 $F^*(t)$灰色长方形的面积来近似计算。

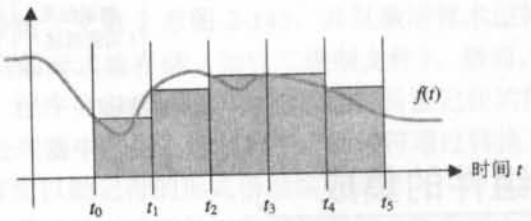


图 4-36　欧拉积分过程

函数 $f(t)$ 的积分结果：

$$F^*(t_n)=\int_{t_0}^{t_n} f(t)\mathrm{d}t$$

通过如下的总和来近似：

$$F^*(t_n)=\sum_{i=0}^{n-1}(t_{i+1}-t_i)\cdot f(t_i)$$

间隔$(t_{i+1}-t_i)$被叫做步长 $\mathrm{d}T_i$。依靠计算任务是否以固定或者可变的激活频率来激活计算过程，步长 $\mathrm{d}T_i$ 是常量或者变量。$F^*(t_{i+1})$可以通过如下的方程以增量的方式计算得到：

$$F^*(t_{i+1})=F^*(t_i)+\mathrm{d}T_i\cdot f(t_i)$$

设计和实现软件组件时，除了考虑近似误差和输入数据的误差，还应特别注意舍入误差。在第 5 章 5.4.2 节详细描述处理器内的数字表示法和在处理数值的过程中出现的舍入误差。

4.7.4　实时模型的设计和实现

设计和实现软件组件的实时模型，必须以深入理解微控制器的硬件和软件中断系统为前提条件。如果应用实时操作系统时，必须确定其配置。实时操作系统的主要配置可以根据第 2 章 2.4 节介绍的 OSEK 标准来设置。

4.8 软件组件的测试

在图 3-28 中描述了保证软件质量的几种方法。以在功能描述和设计阶段指定的测试案例为基础，可以对软件组件进行多种静态测试（如图 4-37）。

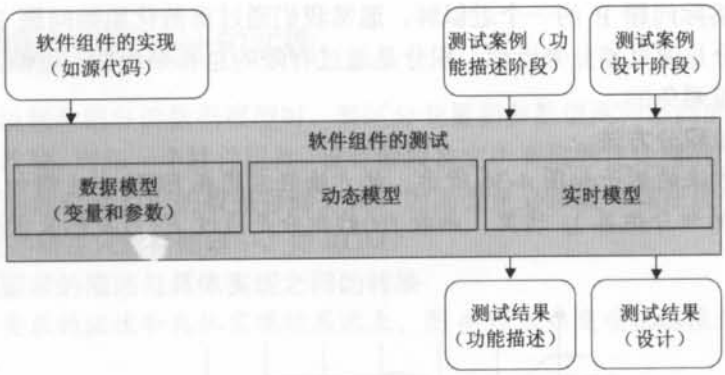


图 4-37 软件组件的测试

4.9 软件组件的集成

由不同的合作伙伴开发软件的不同组件，在微控制器中把这些软件组件组装成整个程序和数据版本的过程称为集成。在集成过程中，要求程序版本和数据版本以符合处理器运行要求的格式产生并提供相应的文档。对于使用的辅助工具（通常在生产和服务阶段中通过外部接口连接到ECU中的微控制器上），集成过程也必须产生合适的描述文档（如图 4-38 所示）。

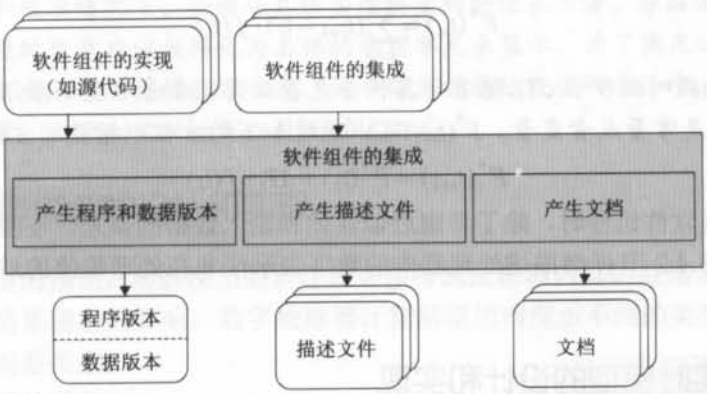


图 4-38 软件组件的集成

因而，产品 ECU 的软件版本通常包括如下：

（1）ECU 的所有微处理器的程序和数据版本；

（2）文档；

（3）用于生产和服务工具的描述文件（如用于诊断、软件参数化和 Flash 编程的工具）。

开发 ECU 可能需要用于 ECU 开发工具的附加描述文档，如：

（1）测量和校准工具的描述文档；

（2）车内通信的网络开发工具的描述文档；

（3）如果使用快速原型工具，还需要所谓的旁路接口的描述文档。快速原型工具将在第 5 章 5.3.8 节中讨论。

4.9.1　程序版本和数据版本的产生

图 4-39 显示产生程序版本和数据版本所需要的处理步骤。在微处理器中任何可执行的命令（机器指令）都是以二进制的数字编码形式来实现的。数字代码（也叫机器码）通过微处理器的控制逻辑来分析（见第 2 章图 2-14），并且激活算术逻辑单元。因而可执行程序必须以二进制机器代码的形式被存储（如以二进制文件）。然而，因为二进制代码的指令比较麻烦，容易混淆，程序员容易出错，因而需要容易被记住的简称（助记符）对应每个机器指令。为了在微处理器中产生二进制程序，助记符通过转换工具转换成机器代码，即编译。因而，源程序首先以助记符的形式借助编辑器书写（如在 PC 机上），然后转换成机器代码。即使在今天，简单的、或者针对特定硬件的、或者对于实时性要求很高的应用场合，仍然（至少是部分）以汇编代码来编程。

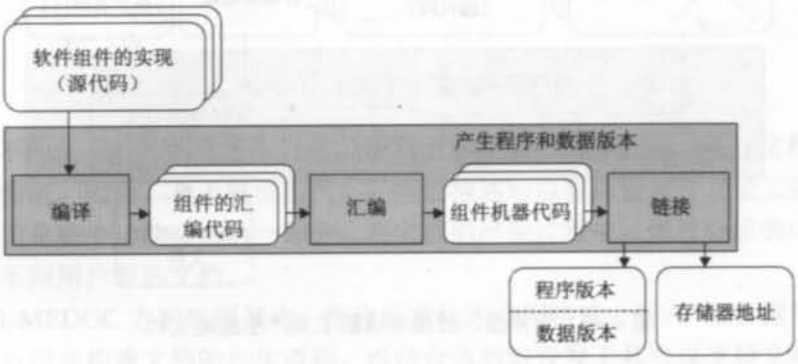


图 4-39　程序版本和数据版本的生成

更复杂的程序需要应用许多更高级的语言，例如 C 编程语言。否则，比较复杂的程序结构将不再清晰、直观，也容易出错，难以维护。因为高级语言独立于微处理器，需要用编译工具（编译器）把用高级语言编写的所谓的源程序编译成针对具体处理器的汇编代码。每个高级指令将被转换成一系列机器指令序列。为此，需要以下类型的机器指令：

（1）数据处理指令：算数、逻辑和转换指令；

（2）控制命令：比较和跳转指令；

（3）输入输出命令：读/写数据指令以及输入输出指令；

（4）存储指令：存储器的读/写指令。

高级语言源代码能以模块且独立于微处理器的形式实现，通过与具体处理器对应的编译器把原代码组件编译成适合各自微处理器的汇编代码组件。

为了建立以机器代码形式存储的不同程序组件间的相互协作和引用，这些组件必须被集成，或者“结合在一起”来形成程序版本和数据版本，这是连接程序的一项功能。连接程序为所有组件分配寄存器地址，并转化为机器代码组件的存储存取地址，连接器程序也把地址信息写进各自的文档。

在许多情况下，编译器、汇编程序和连接器以及其他工具一起被打包装进编译器工具箱。关于相关问题的详细讨论，参考相关的专业文献。

4.9.2 描述文件的产生

程序版本和数据版本，以及用于外部工具的描述文件之间数据一致性的维护，构成了最基本的要求。为此，产生与具体软件部分相对应的描述文档的汇编构成软件集成阶段的一部分。

图 4-40 描述了根据 ASAM-MCD2 标准，为测量、校准和诊断工具产生描述文件的典型过程。

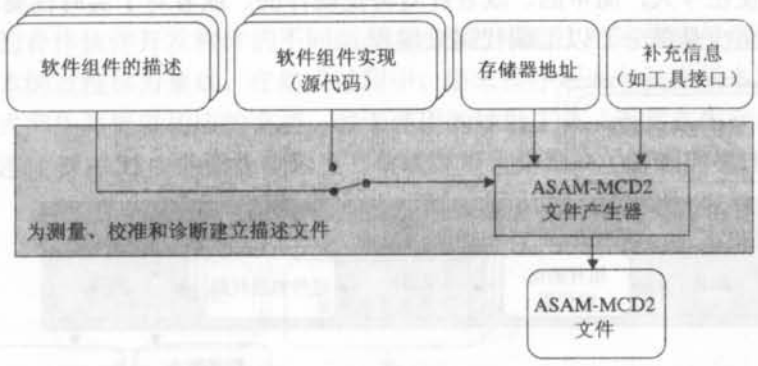


图 4-40 为测量、校准和诊断工具产生描述文档

因为软件组件的描述不仅作为设计和实现的基础而且作为创建描述文件的基础，程序版本、数据版本和描述文档的一致性实际上能够被保证。

作为选择，描述文件所需要的软件组件的描述信息能在实现期间被存储，即以注释形式存储在源代码中。

ASAM-MCD2 文件产生器应用软件组件描述信息和设计信息产生描述文档，并不需要进一步的附加输入，两种方法都要从由连接器产生的文档中提取必要的地址信息。

4.9.3 文档的产生

需要产生由软件来实现的汽车功能的文档，主要有以下几个原因：

（1）在软件开发过程中，所有的外部支持进程都需要文档。另外，例如经常的跨公司的合作，很长的产品生命周期，以及相应的很长时间的软件维护工作，都需要详细的文档。

（2）下一步的开发步骤，例如系统集成、测试和校准都需要文档。

（3）对于汽车的生产和在全球范围内的服务，需要文档的支持。

（4）交通管理部门所需要的文档，例如，汽车文档是申请汽车上路许可证的重要组成部分和前提条件。

毫不奇怪，不同用户群需要的文档有很大的区别（见图 4-41）。因为不同用户群对文档需求的差异，以及对技术基础和细节了解的不同，更不用说不同的语言版本，因而对所有用户群都用一个标准文档是完全不现实的。

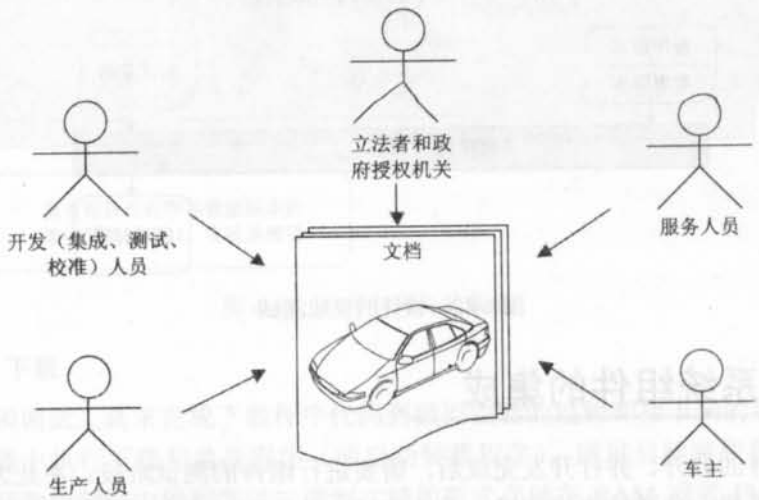


图 4-41　软件功能文档的用户群

然而，不同的文档在面向汽车功能的观点方面是一致的。因此，允许文档结构依据汽车的功能来构造。因而，基于模型的汽车功能的描述可以被用做软件功能文档的基础。

软件文档是整个功能文档的一部分。在文档的产生过程中，通过合适的中间格式，可以构成适合不同用户群的文档。

在 MSR-MEDOC 方案的框架内，作为标准化的中间格式，引入 MSR 报告格式。这个中间格式可以用来构建文档的产生进程，再结合适当的开发工具可以支持文档的产生，如图 4-42 所示。

4.10　软件集成测试

集成软件组件形成软件版本的过程通常需要结合一系列的测试和验证步骤（图 4-43）。这些测试步骤可以在编译前，通过手工来执行，或者通过运行相应的软件工具来自动执行，例如编译器工具箱。测试通常包括检查是否符合接口规范或变量命名的惯例，以及应用统一的存储器分配方案。实际上，因为在测试期间程序本身不执行，所有的测试程序是针对软件实现规范和指南，而进行的静态测试。

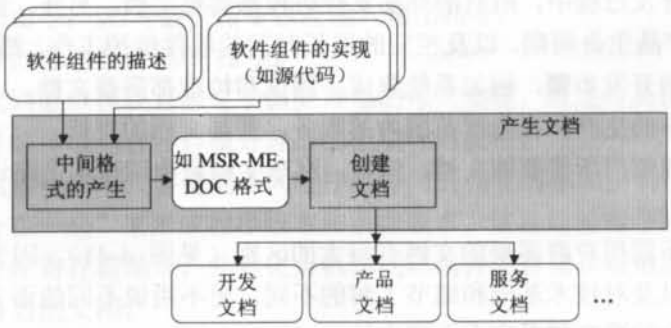


图 4-42　用中间格式生成文档



图 4-43　软件的集成测试

4.11　系统组件的集成

在系统组件的分开、并行开发完成后，需要进行组件的测试阶段。在此之后，把系统的组件集成为系统，并通过集成测试、系统测试和验收测试来完成系统的测试工作。在汽车的各个层次系统的开发阶段，即从单个组件到子系统，再到整个汽车系统（见图 3-4）都要进行类似的工作步骤。因而，实际上集成和测试的概念不能分开。

至于软件，意味着首先与硬件集成（例如与微控制器、ECU 或者实验系统的集成）。

紧接着，将集成不同的 ECU 或实验系统，和设定点发生器、传感器以及执行器，并测试整个系统与被控对象之间的相互作用。所有其他进一步的集成属于更高层级的集成，例如汽车子系统层面的集成或汽车层面的集成（如图 4-44 所示）。

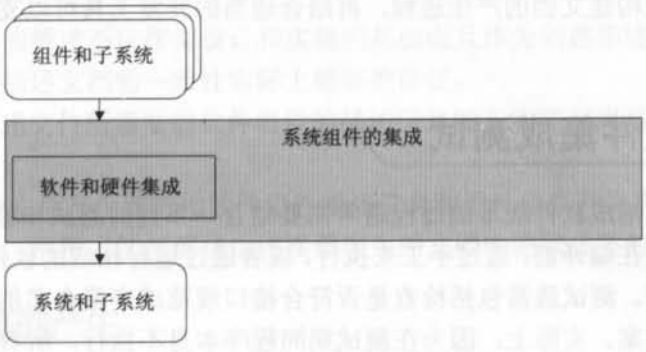


图 4-44　系统组件的集成

4.11.1　软件和硬件的集成

程序的执行和动态测试方法的应用，只有在硬件和软件的集成完成后才有可能。在早期阶段，为了进行这些动态测试，可以应用多种方法。例如，结合实验系统的快速原型方法。软件和硬件集成方法的概况如图 4-45 所示。

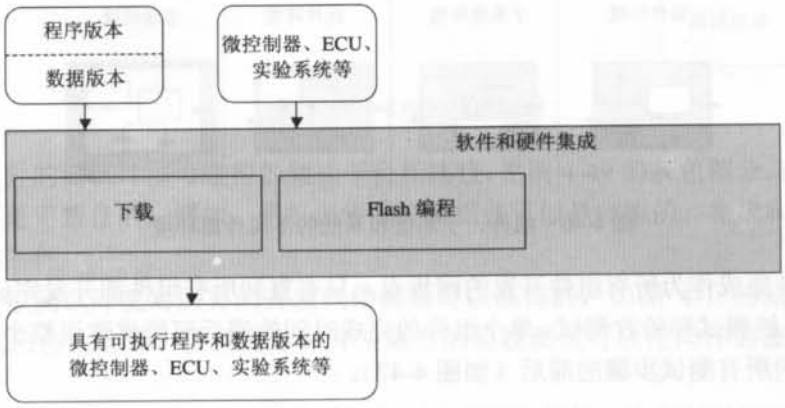


图 4-45　软件与硬件的集成

4.11.1.1　下载

通过下载和调试工具来完成下载程序代码到微控制器的过程和接下来的启动工作。为此，在微处理器中执行下载和监视程序（即启动装载程序）。通过启动装载程序和串行下载接口，下载到微控制器中的程序以二进制文档的形式存储在 RAM 或者 Flash 存储器。根据下载工具的请求，启动装载程序也能提供下载过程的返回数据。

4.11.1.2　Flash 编程

为了对 Flash 存储器的编程，在微控制器中必须有编程程序，并通过 Flash 编程工具来支持。应用这种技术，例如，可以对安装在车内 ECU 的软件进行更新。为此，需要注意的是，必须防止疏忽删除 Flash 编程程序本身所处的存储区域。ECU 的 Flash 编程将在第 5 章 5.6 节和第 6 章 6.3 节进行详细的讨论。

4.11.2　ECU、设定点发生器、传感器和执行器的集成

由于系统组件在多个公司之间分工开发完成，例如分开完成 ECU、设定点发生器、传感器和执行器。因此，对于汽车电子系统的集成和测试工具提出了一系列的特殊要求。

（1）如图 4-1 中描述的测试步骤通常也表示汽车制造商对于供货商提供的组件或者子系统的验收测试。

（2）在开发的过程中，原型汽车的数量是有限的。许多情况下，提供组件的公司不具有所提供的组件所处的完整或真实的环境。这些环境对于每个组件是不同的（如图 4-46 所示）。由于测试环境的限制，部分供应商倾向于减少一些对于组件的测试程序。

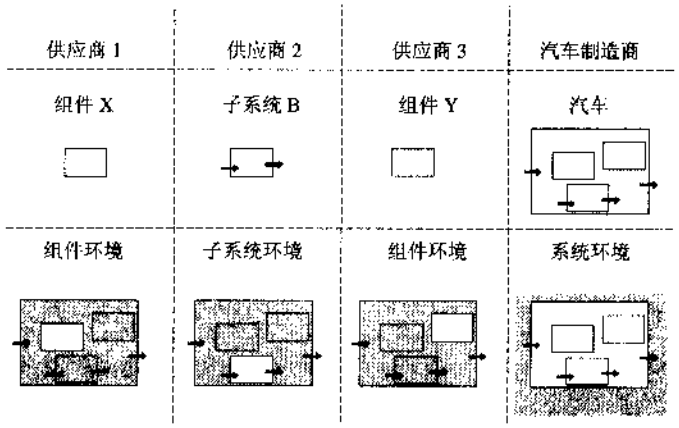


图 4-46 组件、子系统和系统的不同外部环境

（3）组件集成作为所有组件开发的同步点，只有直到所有组件都开发完成，才能执行集成测试、系统测试和验收测试。单个组件的完成时间的滞后可能将耽误整个系统的集成，并引起随后的所有测试步骤的滞后（如图 4-47）。

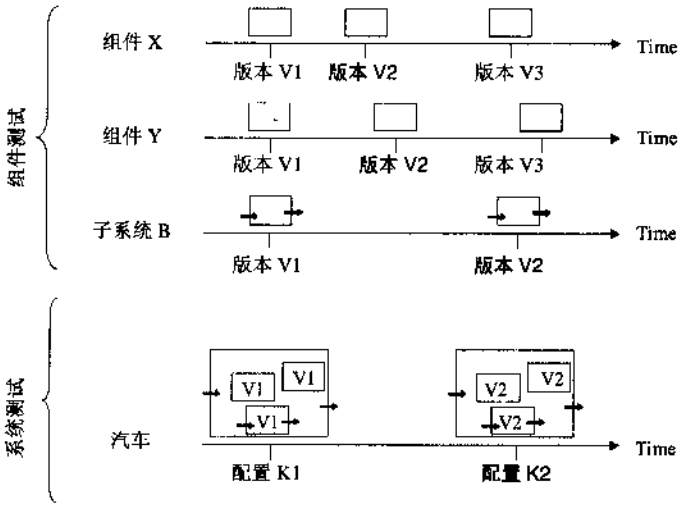


图 4-47 组件测试和系统测试的依赖关系

4.12 系统的集成测试

汽车系统的集成和测试工具考虑到前面章节讨论的对于集成和测试工具的特殊条件，并减少系统组件间相互依赖性和开发危险。测试可以通过合适的支持工具自动进行，图 4-48 表示系统的集成测试的前提条件和结果。

可以获得的组件、子系统和系统环境组件作为真实的组件而被集成。不存在的组件、子系统和系统环境组件而用建模和仿真的方式来模拟（例如虚拟组件）。

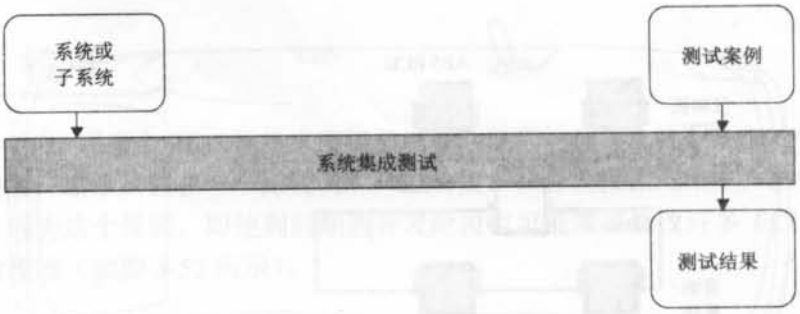


图 4-48　系统的集成测试

真实组件的测试环境与虚拟的集成平台相连接，见图 4-49 的灰色部分。在这个环境下，任何组合都是可想象的。例如，系统组件或环境组件可以是虚拟的。集成和测试工具必须满足下列的条件：

（1）虚拟测试环境对所有的开发伙伴都是可以获得的，如图 4-1 的测试步骤可以作为验收测试来自由选择，供应商方和开发商方的验收测试可以在同样的虚拟测试环境中进行。

（2）每一个合作伙伴都有机会获得同样的全套虚拟组件，他们可以配置这些虚拟组件来构建适合自己的测试环境。

（3）因为系统测试也可以用虚拟组件代替不存在的真实组件，因而单个组件的延迟而耽误集成的风险减少（如图 4-49（c 和 b））。同样，真实组件和虚拟组件的组合也是可能的。

（4）最初完成的系统环境可能全部是由虚拟组件来构成，这些虚拟的组件可以逐步用真实组件代替，逐步代替可以出现在系统的各个层面上。

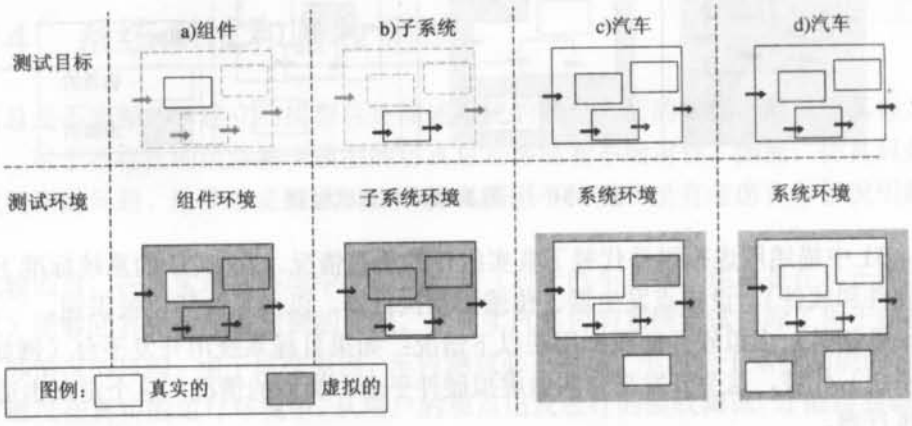


图 4-49　测试对象和测试环境

举例：仪表盘的虚拟网络环境

图 4-50 描述了仪表盘虚拟网络环境的组件，这种测试环境也叫做剩余总线仿真。系统的功能模型可以作为构成不存在的虚拟系统组件的基础。

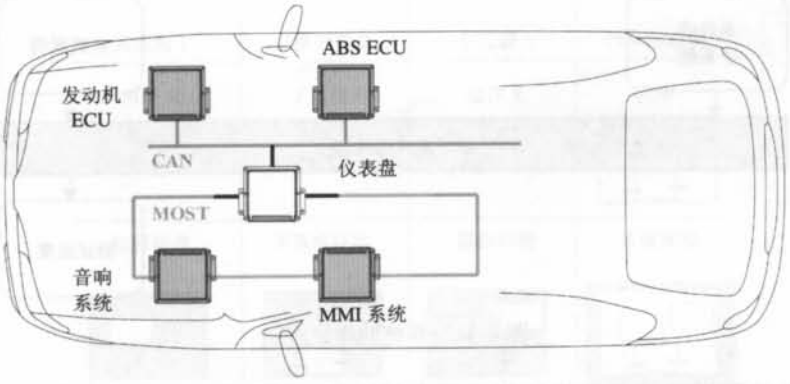


图 4-50 仪表盘的虚拟网络环境

这种集成和测试过程带来了其他额外的优点：

（1）没有测试环境时，许多测试程序将需要在汽车内运行，现在可以搬到实验室汽车或者测试平台上运行。

（2）与在道路上行驶的真实汽车上的测试相比较，这种方法提高了测试和应用实例的可重复性。此外，也易于实现测试的自动化。

（3）采样这种测试方法，在测试极端情况下，对测试驾驶员和原型汽车没有任何的危险。

一致的集成和测试过程：从虚拟仿真、在实验室汽车上的测试、试验台测试，直至在真实汽车中的测试，因此而成为可能（如图 4-51）。

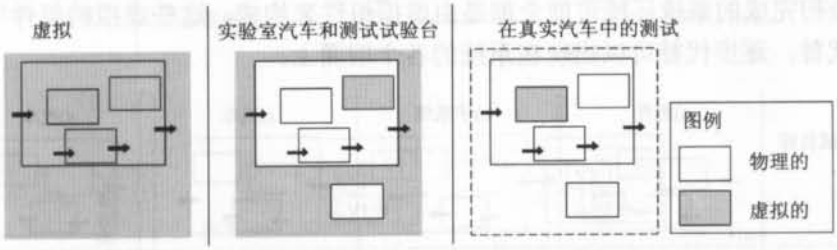


图 4-51 一致的集成和测试过程

图 4-51 中描述用虚拟组件代替了真实组件的所有情况。在 ECU 的系统标准下，组件 ECU（硬件和软件）、设定点发生器、传感器和执行器，以及系统环境被识别。

与硬件相关，虚拟硬件的概念用在以下情况：如果目标系统用开发平台（例如 PC 或者实验系统）代替，这个开发平台叫做虚拟硬件平台。类似的情况，这个定义也适用于传感器和执行器。

当真实的硬件和软件都开发完成时，才可以运行功能程序。然而，许多测试程序只有与传感器、执行器和系统环境集成后才可能执行。集成和测试工具将在第 5 章 5.5 节讨论。

成功完成测试步骤后，下一步是发布程序版本和数据版本。然而，数据版本仍然可能在后期开发阶段的校准过程中被修改，而程序版本保持不变并且被发布。

4.13　校准

通过单个 ECU 或整个 ECU 网络来实现的软件功能的协调，也就是汽车的软件功能的参数调整与设置，通常只能在一个比较晚的开发阶段，并且直接在汽车中、系统在运行的情况下进行。因为这个原因，即使到后期的开发阶段也可能需要修改许多 ECU 的数据版本。这步叫做校准（如图 4-52 所示）。

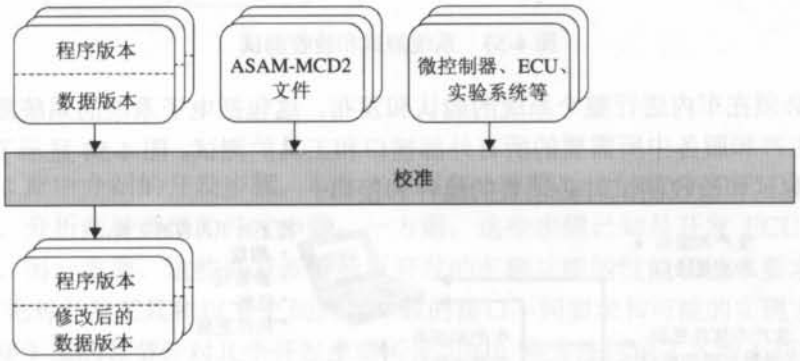


图 4-52　校准

校准在后期开发阶段通过应用校准系统来完成。校准系统必须能修改数据版本，数据版本固定存储在只读存储器内，例如 ROM、EEPROM，或者闪存。因此，校准系统由具备与测量和校正工具连接的外部接口功能的 ECU 构成。各种校准方法详见第 5 章 5.6 节。经校准阶段以后，给下一开发步骤提供数据版本，如图 4-1 所示。

4.14　系统测试和验收测试

建模总是不完整的。建立的模型总是描述系统的某个方面的特性，而忽略其他方面特性。为此，对于不存在的组件基于模型的仿真总是伴随着不确定性。因而，仿真只是回答实际中被问到的问题。结果总是要承担由于仿真模型不精确和没有考虑实际情况引起的那些风险。

相比较而言，在真实运行环境中的测试步骤（对于汽车系统，也就是在真正的汽车中进行测试）能够回答先前没有问到的问题。在真实汽车内的系统测试排除了由于建模缺点而忽略实际情况的风险。由于这些原因，车内测试必不可少。车内电子系统功能的有效性，最终只有通过在真实的运行环境中，从用户的观点出发进行的验收测试，才能得到验证（如图 4-53 所示）。

系统测试和验收测试对开发方法论和工具提出了特殊要求，例如，支持与车内 ECU 和 ECU 网络的通信，移动测量技术的应用，在恶劣环境中的运行的配置，以及汽车内运行和图形的可视化等。

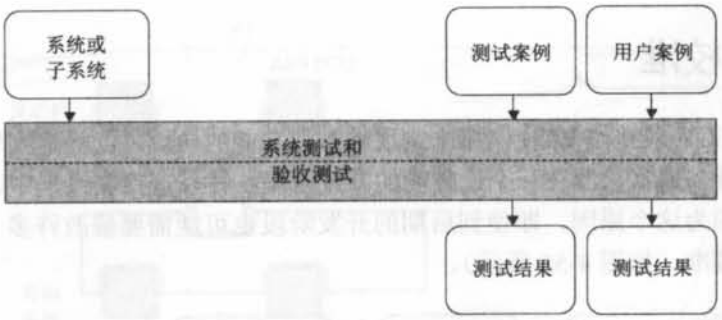


图 4-53 系统测试和验收测试

最后，必须在车内进行整个系统的确认和发布。这包括电子系统的系统测试和验收测试，以及在生产和服务中所需要的所有外部接口和工具的测试。图 4-54 显示了仪表盘系统在进行系统测试和验收测试时必须要的组件和接口。

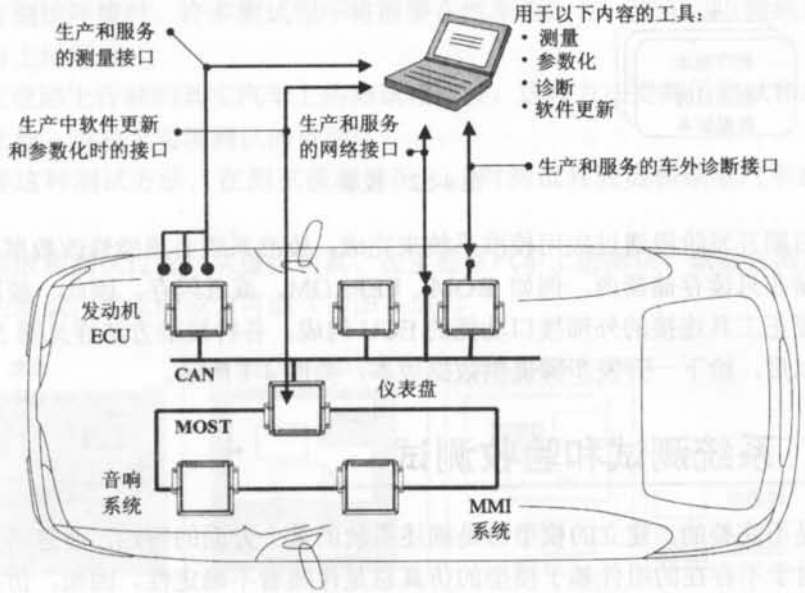


图 4-54 汽车仪表盘的系统测试和验收测试时的接口

第 5 章 Chapter 5

开发方法与工具

沿着第 4 章中介绍的开发步骤，本章将介绍整个车辆功能的软件实现工过中所使用的方法和工具。分析两种类型的开发步骤：一方面，这些步骤已知是开发 ECU 应用软件功能所必需的，另一方面，这些开发步骤给所开发的车辆功能的性能和技术要求提供保证。

5.1 节首先对开发工具和ECU之间的非车载的接口不同要求和可能的实现方法进行了初步的概述。接下来的各节针对几个开发步骤所采用的一些方法讨论了各种不同方法和工具。

实际应用中的系统和软件开发必须满足高度的安全性和可靠性要求，也就是车辆在每个开发阶段必须要求有系统和软件质量控制标准。本章对这一要求给予特别研究。

5.2 节研究了逻辑系统结构分析和技术系统结构描述所使用的方法。

（1）开环和闭环控制系统的分析和描述；

（2）实时系统描述和可调度性分析；

（3）分布式和网络化系统中通信的分析和描述；

（4）可靠性和安全性分析，以及可靠性和安全性概念的描述。

为了实现软件功能的描述及验证功能描述的有效性，可以采用基于模型的方法。这样做除了可以给技术要求提供一个非二义性的、精确的系统描述外，还能促进一个给定软件功能的早期验证。为此，5.3 节讨论一些合适的方法，例如：

（1）形式描述和建模；

（2）仿真和快速原型化。

5.4 节主要研究软件功能设计和实现的方法和工具。因为这涉及到要把一个描述映射为具体的算法，所以必须留意所要求的非功能性的产品特性。这些包括：

（1）根据所要求的硬件资源，软件开发中的优化方法；

（2）利用程序语言的有限子集提供严格的可靠性和安全要求，一个恰当的例子是 MISRA-C Guidelines [80]；

（3）通过标准化和软件元件的重用降低质量风险。

5.5 节精选了一些软件功能集成和测试方法进行讨论。这些包括：

（1）与开发并行的测试，例如不同系统级上的元件测试、集成测试或系统测试；

（2）实验室中、测试台上以及车辆中集成、系统和验收测试。

5.6 节对软件功能校准方法和工具进行探讨，它包含以下内容：

（1）微控制器和所谓的在线校准工具间的接口；

（2）在真实汽车中的软件功能的测量和校准过程；

所述方法和步骤的实际意义将通过汽车动力系统、底盘和车身应用领域的例子来说明。

5.1 工具和电子控制单元之间的非车载接口

整个开发过程不同阶段所采用的众多工具与 ECU 微控制器之间需要一个非车载的接口。常见的应用工具主要有：

（1）程序下载和调试工具；

（2）Flash 编程的软件更新工具；

（3）ECU 网络接口的开发和测试工具；

（4）快速原型化工具；

（5）开发 ECU 的测定和校准工具；

（6）ECU 的参数化工具；

（7）非车载的诊断工具。

一个给定工具与对应的 ECU 之间的接口必须有工具和 ECU 两方面的硬件和软件组件的支持。在不同的开发阶段，非车载接口必须满足不同技术要求。对于接口要求的一些例子如下所示：

（1）实验室中以及处于严酷环境的车辆中的使用；

（2）在微处理器程序执行中断或不中断的情况下，工具与微处理器的通信；

（3）接口传输速度的不同需求；

（4）仅在开发中应用，或也能在生产和服务中应用；

（5）工具与 ECU 之间的非车载接口，是否需要把 ECU 从车中拆除下来。

任何车辆电子系统的开发都以生产许可及后续的服务来结束。对于 ECU 的开发，这意味着开发结束时的验收测试必须使用与生产以及服务中同样的非车载接口和工具。

车辆应用的非车载接口的技术要求在许多方面和实验室环境制造的接口存在根本差异。例如，车辆应用对温度范围、冲击以及振动、电源稳定性和电磁兼容性有很高要求。而且，车辆上 ECU 的安装位置会使非车载接口的安装空间受到约束。此外，ECU 及其对应的工具间通常具有更大的空间连接距离。

因此，开发过程中要使用不同的接口技术。图 5-1 所示为与设计非车载接口有关的微控制器部件的概貌。这些部件是微处理器、内部和外部 ROM、内部和外部的闪速内存以及 RAM、微控制器内部和外部的总线，还有微控制器本身不同系列的串行接口。

5.6 节将讨论软件功能标准的特殊要求以及对应的非车载接口的构造。另外，生产和服务中所采用的非车载接口的详细信息将在第 6 章讲述。

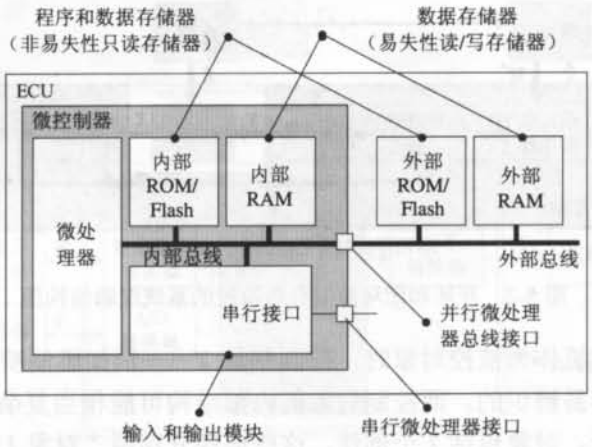


图 5-1　微控制器接口

5.2　逻辑系统结构分析和技术系统结构描述

开发步骤的第一步是定义逻辑系统结构，也就是以信号形式表示的功能网络、功能接口，以及整车或车辆子系统功能之间的通信。

第二步在一个具体的技术系统结构中构建抽象的逻辑函数功能。这一步受到参加开发的不同工程领域的分析和描述方法的支持。一方面，这有助于技术可行性的早期评估，另一方面可以对不同的实现方案进行对比、比较和评价。接下来的几节将介绍影响软件功能实现的分析和描述方法。

5.2.1　开环和闭环控制系统的分析和描述

仔细观察一下许多车辆功能的本质，你会发现它们其实表示开环或闭环控制功能。因为所有类型的控制功能越来越依靠软件来实现，因此依赖于数值仿真工具的控制技术的分析和设计方法对许多软件功能的开发产生了重大影响。

本节不打算详细讨论各种分析和设计方法。有关这方面的讨论可参阅参考文献[34，35]。本节主要讨论车载控制功能分析和设计的早期阶段必须考虑的准则，只要上述功能是基于软件和电子系统实现的。

解决控制问题与被控对象的具体结构无关。然而对象的静态和动态特性却是最重要的。因此，控制技术分析方法的第一步是被控对象的分析。一旦对象的系统边界被建立，对象的输入输出参数（I/O 参数）就被定义了。描述部件输入输出间静动态关系的物理模型方程通常采用辨识方法建立，所得到的对象模型成为所有后续设计程序的基础。

完成一个给定控制功能任务的电子系统的所有部件与控制系统所需的功能部件（例如设定点发生器、传感器、执行器以及 ECU）联系起来。在早期开发阶段，通常作法是将系统简化成部件、接口、以及它们之间的相互关系，如图 5-2 所示。控制系统具体的技术结构设计仅在技术系统结构设计中才被定义。

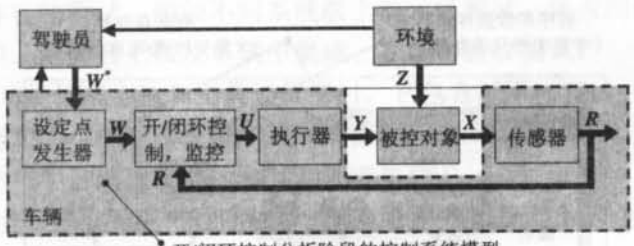


图 5-2 开环和闭环控制分析阶段的系统逻辑结构图

例如，当把内燃机作为被控对象时，控制变量 ***Y***——例如燃油喷射量、点火定时、节气门开度等，都是容易辨识的。而控制对象的内部结构可能相当复杂，图 5-3 给出了一个比较简单的对象例子。对象包括 7 个部件，这些部件分别用“对象 1”到“对象 7”标志，这是设计逻辑控制系统结构的基础，图 5-3 提供了 7 个控制器（分别用“控制器 1”到“控制器 7”表示）。

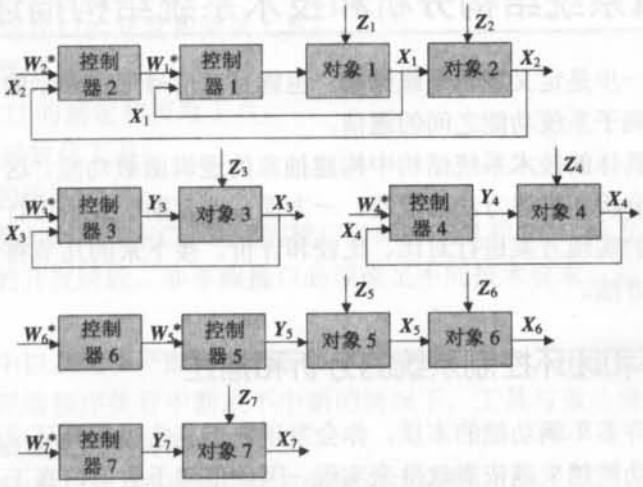


图 5-3 开环和闭环控制系统的逻辑系统结构

接下来的步骤是控制系统各控制部件的控制策略的设计，最后是设定点发生器、传感器、执行器、ECU 以及它们的软件功能的技术系统结构设计。图 5-4 描述了控制器 3 的技术系统结构，软件功能的描述方法在 5.3 节讨论。

控制系统的控制策略设计这一步中，值得注意是：在许多车辆应用中，组成一个电子系统的部件的传递函数并不是理想的。考虑到成本，设定点发生器、传感器、执行器和硬件模块通常只提供有限的分辨率和动态。另外，必须考虑微控制器的时间离散和值离散的工作方式。在一个控制系统设计阶段的早期，必须考虑设定点发生器、传感器、执行器以及微控制器 A/D 和 D/A 转换器的以下特性：

（1）由于分辨率有限，离散值运算所产生的影响；

（2）非线性，比如条件约束所导致的；

（3）受限动态导致的时滞或死区时间。

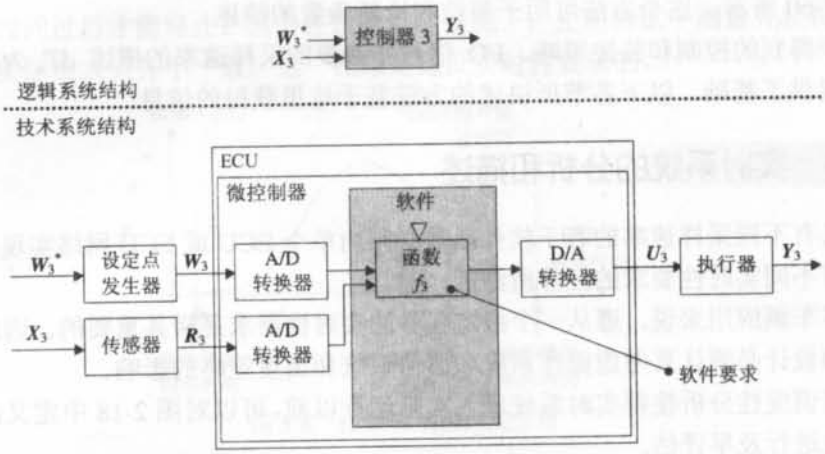


图 5-4　控制器 3 技术系统结构设计

在许多情况下，微控制器有限的硬件资源会对软件功能的实现产生不利影响。因此，下列内容必须考虑：

（1）由四舍五入，或处理下溢或溢出所导致的误差，例如当采用整数运算时；

（2）逼近误差，例如由算法精度有限导致；

（3）微控制器离散时间控制所产生的影响。

对象的时间常数决定控制系统的采样速率 dT，和确定软件函数 f_n 的采样速率 dT_n。



图 5-5　软件函数 f_n 外部视图

图 5-5 为开发阶段一个软件函数的外部视图。

图 5-3 中的控制部件也可以像图 5.6 那样在一个 ECU 中来实现。7 个控制部件（控制器 1 到控制器 7）可通过采用设定点发生器、传感器、执行器、A/D 和 D/A 转换器还有软件函数 f_1 到 f_7 来实现。

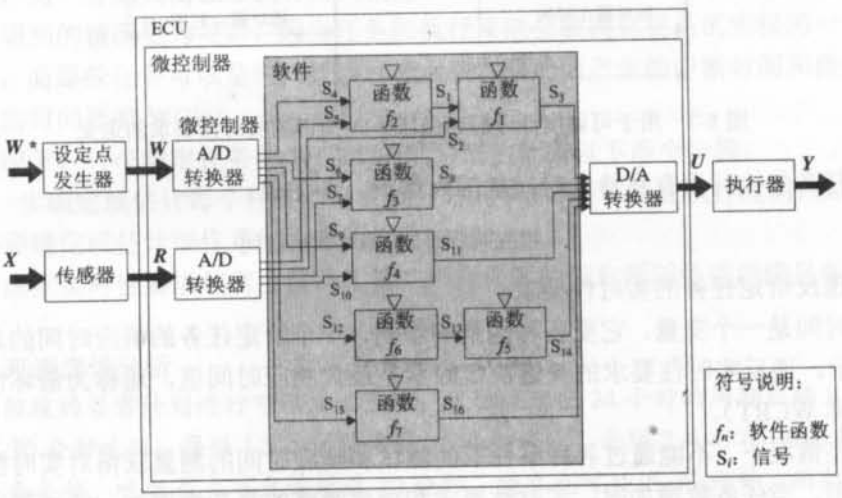


图 5-6　开环和闭环控制系统技术系统结构设计

如图 2-60 所示，这个方法可用于监控和诊断函数的描述。

这一步得到的控制和监控策略、I/O 信号和必要的采样速率的描述 dT_n 为所有软件函数 f_n 开发提供了基础。以下各节所讲述的方法基于这里获得的信息。

5.2.2 实时系统的分析和描述

因为具有不同采样速率的若干软件函数必须由单个 ECU 或 ECU 网络实现，所以软件函数被具有不同实时性要求的任务所激活。

对许多车辆应用来说，遵从一个给定任务的实时性需求是极其重要的。因此，实时系统的分析和设计必须认真考虑操作系统和通信系统和调度策略的影响。

应用可调度性分析使得实时系统进入实际运行以前，可以对图 2-18 中定义的实时性要求的适应性进行及早评估。

这里我们可以区别处理器任务调度分析和通信系统节点间总线判优分析，这两项任务所采用的方法非常类似。本节讨论利用处理器调度的可能方法。然而，在实际应用中，这些分析方法可由合适的设计、验证和监控原理来补充。由此可得到实时系统的描述，其中所有软件函数已被划分为过程，这些过程被分配给任务。

没有一般适用性的限制，首先假设实时系统每个任务的实时性要求的标准化定义建立在以下基础上：

（1）一个给定任务的两次激活间定常或变化的实间间隔用图 5-7 中的“激活速率”表示；

（2）从任务的激活点开始至任务执行完成的时间间隔，被称为相对截止时间。

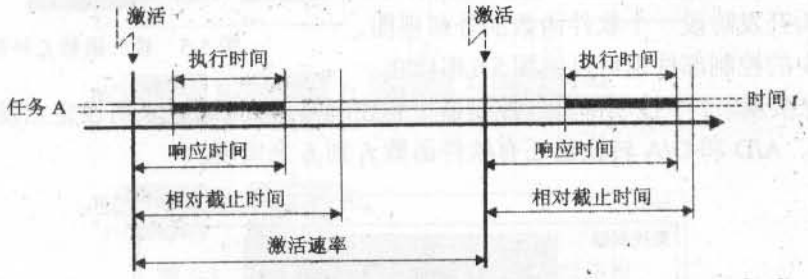


图 5-7 用于可调度性分析的以任务 A 为例的实时性要求的定义

如果任务的执行没有在特定时间范围内结束，即如果：

$$\text{相应响应时间} > \text{相对截止时间} \tag{5.1}$$

那么必然违反给定任务的实时性要求。

响应时间是一个变量，它受各种因素的影响。一个给定任务的响应时间的典型分布如图 5-8 所示。违反实时性要求的关键决定因素是最大响应时间值，通称为最坏情况响应时间（或简称 WCRT）。

在某些情况下，不能通过各种条件下的测试和响应时间的测量获得对实时性要求是否满足的证明。当任务数增加时，实时性要求和调度策略的复杂性增加，通过测试获得这种证明实际上通常是不可能的。甚至在“成功”测试之后，在临界情况下，一个任务的执行

只有在截止时间过后才能终止。因为在这种临界情况下，先期测试中测量并获得的最大响应时间值与实际最大值并不一致，这一结果是违反实时性要求的。

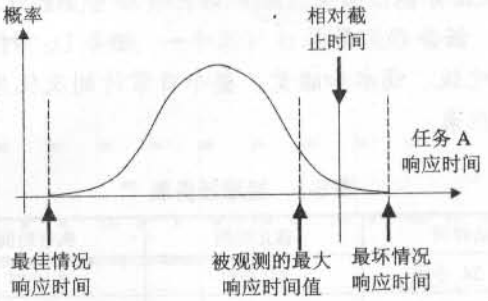


图 5-8　任务 A 响应时间分布

因此，实际应用中常采用三种方法的组合：

（1）可调度性分析以评价各种实现方法；

（2）通过测量来验证可调度性分析的结果；

（3）通过实时操作系统在线截止时间的监测，以及对违反截止时间的具体响应和处理。

5.2.2.1　可调度分析

可调度性分析的目的在于利用所有已知参数对实时性要求提供事前评估。

实时系统的要求是这样的：

$$\text{最坏情况响应时间(WCRT)} \leqslant \text{相对截止时间} \tag{5.2}$$

这要求对 WCRT 进行确定或估计。

在图 5-7 描述的简单情况中，响应时间一方面由激活点和任务执行起始时刻之间的时间间隔决定，另一方面又由任务执行时间决定。

通常所遇到的情况更为复杂，因为任务的执行可能会被拥有更高优先权的一个或更多的任务中断，而那些任务可以是时间或事件激活的，由此而产生的中断时间和操作系统任务转换所需的时间影响 WCRT。

正常情况下，一个给定任务的 WCRT 的确定或估计需以下两个步骤。

（1）第一步确定或估计每个任务的最大执行时间（称为最坏情况执行时间，或 WCET）。另外，还必须确定或估计操作系统本身的执行时间。

（2）考虑到实时性要求和调度策略，第二步对任务的所有激活是否能满足条件（5.2）进行估计。

举例：可调度性分析

对一名经理的日常计划进行可调度性检验，这位经理在 24 小时的周期内睡 8 小时，每隔 8 小时花 30 分钟吃饭，每隔 1.5 小时他要花 15 分钟喝水，每隔 2 小时花 30 分钟打电话。

作为一个条件，吃饭最大可允许延迟 30 分钟，喝水也可延迟 30 分钟，而打电话只能延迟 15 分钟。必要的睡眠时间必须在 24 小时内结束，这导致下列截止时间：睡眠=24 小时，吃饭=1 小时，喝水=45 分钟，电话=45 分钟。

假设按照基本任务状态模型如 OSEK-OS（图 2-20）应用优先调度策略，目的是检验这位经理能否处理这些约定。照目前情况看，这位经理必须完成 4 项任务：

任务 A：睡觉　　任务 B：吃饭

任务 C：喝水　　任务 D：打电话

优先级是：打电话、吃饭、喝水和睡觉。整个日常计划及优先权、激活时间、截止时间和执行时间被归纳成一张表：

表 5-1　经理任务表

	激活时间	截止时间	执行时间	优先级（4 为最高）
任务 A	每隔 24 小时	24 小时	8 小时	1
任务 B	每隔 8 小时	60 分钟	30 分钟	3
任务 C	每隔 1.5 小时	45 分钟	15 分钟	2
任务 D	每隔 2 小时	45 分钟	30 分钟	4

可调度性可通过以下执行情况进行检验（图 5-9）：

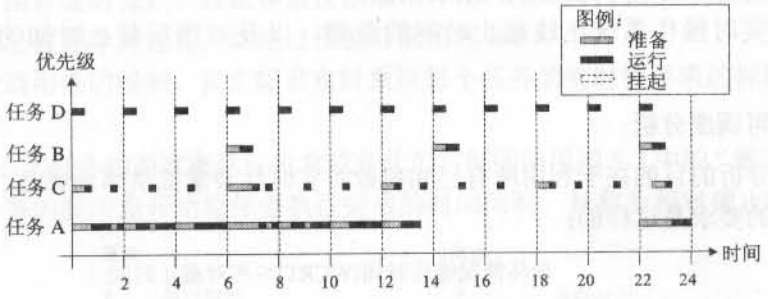


图 5-9　优化前的调度图

（1）任务 D：打电话，具有最高优先级，每隔 2 小时执行一次，不违反实时性要求。

（2）任务 B：吃饭，与任务 D 在 6：00、14：00 和 22：00 同时激活。然而，由于它的优先级比 D 稍低，它的执行延迟 30 分钟。执行在截止时间前正好结束。

（3）任务 C：喝水，每隔 90 分钟激活一次。然而，由于它的优先级较低，执行可被任务 B 或任务 D 中断或延迟。在 4 种情况中，45 分钟截止时间正好满足。最坏情况是在 6：00。在这里，响应时间是 75 分钟，超出了截止时间。这里任务 C 在任务完成后 15 分钟就可被再次激活。

（4）任务 A：睡觉，具有最低优先级，被延迟 75 分钟后启动（23：15），正如所预料的那样，任务被频繁中断，从任务激活到执行结束超过 15 个小时。

任务 C 的临界情况在 6：00，任务 B 实时性要求的临界情况通过采用各种方法可以变得无害。图 5-10 描述了一种情形：其中，任务 B 没有与任务 D 同时激活，而是偏移了 1 小时，即在 7：00、15：00 和 23：00。这保证任务 B 的实时性要求自始至终是安全的。任务 C 在 6：00 的临界情况采用同样方法可以变得易于忍受，且在五种情况的截止时间正好满足。补偿临界情况的另外方法是延长任务 C 的截止时间，比方说，60 分钟。另外可能的做法是在条件允许时提高优先权。可以预料的是，新方法不会影响任务 A，因为它的优先级最低。

现在还可能回答进一步的问题，即以任务 E 的形式作另外进一步安排。如图 5-10 所示，现在 14：00 和 22：00 之间安排另外的日常任务是可能的。为了给我们繁忙的经理安排一个更为平衡的计划，“睡觉”任务可被分割为两个任务“晚睡眠”和“午睡”。可以估计未预料到的更高级的中断，例如，一个客户打给经理的电话。

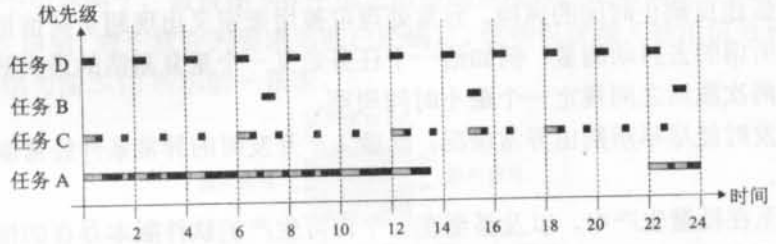


图 5-10　优化后的调度图

尽管我们的例子中这个虚拟经理的情况被刻意抽象，但除了说明优化方法的效果外，它仍然能描述实时系统描述分析上的误差。

除了给现有的每项任务提供可调度性分析的细节，这种分析还便于对计划密度信息进行更为仔细的研究，这些也可以用来改善实时性要求的描述。结果，整个系统的负载得到更均匀的分布。在某种情况下，更均匀的负载甚至可以降低所需硬件资源的配套，例如处理器时钟频率。

理想情况是低优先级的任务可以被更高优先级的任务在任意时刻中断。事实上，实际应用中经常会出现条件限制。就我们的例子而言，这意味着任务“睡觉”可能每隔 2 小时就会被中断一次。

还有，理论上假设任务之间的切换完全没有延迟是不现实的。实际应用中会出现条件限制，在我们的例子中，任务“睡觉”和“打电话”之间转移需要插入延迟或唤醒阶段。

当这种程序方法用于车辆应用中，实时操作系统本身需要的执行时间必须另外考虑。这些可能要依赖于大量的参数，特别是所选择的调度策略。另外一个问题是给定任务最坏情况执行时间的估计，通常它也依赖于各种参数。

要进行执行时间的计算和估计，例如，基于编译程序生成的指令，往往要花费大量时间，且要根据具体情况设置特定的方法。这些仅在某些特定情况下是可能的，这个问题是我们所从事的大量研究工作的主要内容。

在这种情况下，应严格限制一些程序结构，比如处理时间可能任意长的重复环或等待状态。因此，迭代算法应该在保证每个任务只有一个或有限迭代步骤的方式下执行。如果不可能在有限步骤内结束，迭代算法执行时间的最坏情况的估计就不能进行。

5.2.2.2　通过测量验证可调度性

实时行为的验证可通过测量实际系统的激活时间和执行时间以及画判优图来完成。采用这种方法，糟糕的设计缺陷能被识别出来，也能够逐步改善实时系统的参数配置。

尽管如此，但应该记住，如图 5-8 所示，测量或观测到的执行时间仅提供了一个最大执行时间和响应时间的近似值。

此外，在绝大多数情况下，测量时甚至不可能保证一个给定系统真正处于临界负载状态。这种情形在具有变化的任务激活时间和事件（例如中断）的事件控制系统中是典型的。

5.2.2.3 操作系统中违反截止时间的监测和处理

对一个给定的任务，可以通过改变多种属性，比如优先级、截止时间本身或设置一个激活延迟来降低违反截止时间的风险。异常处理可被用来定义出现超负荷情形时的处理行为。这些包括所谓的去抖动测量，例如给一个任务定义一个重复激活的最大数目，或在一个给定任务的两次激活之间规定一个最小时间周期。

为了在开发时能尽早辨别出异常情况，批量生产开发时的异常条件经常制定得比后续步骤更为严密。

然而，甚至在批量生产中，以及甚至在一个许可生产的软件版本存在的情况下，也必须由实时操作系统实行在线截止时间监测，尤其是那些必须满足严格的实时性要求的任务，违反截止时间的处理通过与特定功能对应的错误处理程序来完成。

合适的实现可以利用函数特定的错误分支，在检测到违反截止时间的情况时由操作系统调用。在这种情况下，这些软件监测函数扩大了由硬件实现的监控方法，例如，采用看门狗模块监控程序的执行。

5.2.3 分布式网络化系统的分析和描述

ECU 软件功能的输入参数可以通过直接分配给 ECU 的传感器而获得。另一个可选择的方法是利用分配给其他 ECU 的传感器的信号，这些信号通过车载通信网络传输，它们的使用可被描述为间接使用，对于执行器存在同样的选择方案。另外，所有 ECU 内部计算的信号和状态可以通过通信网传输。

尽管这个结果能给函数设计带来相当大的自由度，但它也会产生新的设计问题，即 ECU 或微控制器网络上软件函数的最有效分布问题，以及传感器和执行器的抽象问题。图 5-11 表明依靠电子系统部件间逻辑通信连接的柔性，而直接或间接带来的柔性和自由度。

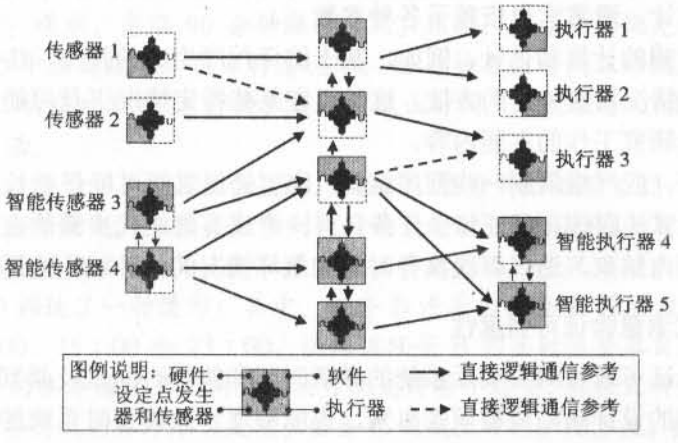


图 5-11 分布式和网络化系统的逻辑系统结构分析

另外，也必须考虑到通信系统的影响，例如消息传输时信号值的离散化，或通信系统的传输时间（响应特性）。这些在设计分布式或网络化系统时应尽早考虑。

分布式或网络化系统的分析方法使我们能够评价这些因素带给系统的影响。与此同时，必须考虑到大量的性能要求和约束限制，例如安装空间、实时性、安全性和可靠性。

作为一个引导例子，假设图 5-6 中描述的控制系统软件函数需要分配给不同的微控制器。第一步是辨别，对于软件的要求那些已经确定。这些包含输入输出信号和采样速率 dT，这些要求归纳为图 5-12 所示的一张表。



函数	dT	信号	输入	输出
f_1	10 ms	S_1 S_2 S_3	X X	X
f_2	20 ms	S_4 S_5 S_1	X X	X
f_3	10 ms	S_6 S_7 S_8	X X	X
⋮	⋮	⋮	⋮	⋮

图 5-12　带采样速率和信号的软件功能的图表描述

为了把这些软件函数分配给不同的微控制器，这张表中加入了新的一列，如图 5-13 所示。该列中的信息表示将要被执行的函数的具体位置。函数 f_1 在微控制器 μC1.1 上计算，函数 f_2 在微控制器 μC2.1 上计算，函数 f_3 在微控制器 μC1.2 上。在分配中，必须考虑现有的硬件结构条件。已有的传感器和执行器与特定的微控制器之间的对应关系限制了软件函数在微控制器上的分配。例如，预处理传感器信号的软件函数直接分配给管理这个传感器的微控制器。

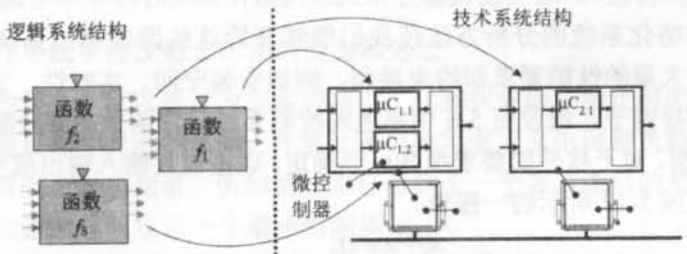
下面的讨论，区分两种情况：

（1）如果信号不仅重复发生而且发生在图 5-13 所示表格中的不同微处理器上，那么必须通过网络进行通信。因此可以获得通过通信系统传输的所有信号的集合。在图 5-13 的简化例子中，这仅仅应用于信号 S_1，S_1 必须从微控制器 μC2.1 传递到微控制器 μC1.1。

（2）然而，如果信号重复发生在图 5-13 所示表格中的一个给定微控制器上，且具有不同的采样速率，那么这些信号必须在任务间进行通信。因此，可以获取必须通过任务间相互通信传输的所有信号的集合。

在第一种情况下，我们可能会立即提出这样的问题，信号必须以多大的速率通过网络进行传输。在我们的例子中，函数 f_2 以 20 ms 的采样速率计算信号 S_1，因此以更快的速率传送信号 S_1 几乎没有意义。即使如图 5-13 所描述的那样，以 10ms 这样更快的速率计算接收函数 f_1，上述结论也是对的。在这种情况下，所产生的规范是以 20ms 的采样速率传送

信号 S_1。



函数	dT	信号	输入	输出	微控制器	传感器	接收器
f_1	10 ms	S_1	X		$\mu C_{1.1}$		X
		S_2	X				
		S_3		X			
f_2	20 ms	S_4	X		$\mu C_{2.1}$		
		S_5	X				
		S_1		X		X	
f_3	10 ms	S_6	X		$\mu C_{1.2}$		
		S_7	X				
		S_8		X			
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

图 5-13 软件功能在微控制器上的分配

相反，如果以比 f_2 更慢的采样速率计算接收函数 f_1，那么以接收函数的采样速率传送信号将使通信系统的负载减轻。在每种情况下，对于信号传输速率，所需要的传输时间以及传送和接收函数的采样速率 dT，必须加以区分。

以下步骤定义通信系统。为此，需要建立由通信系统传送的消息。换句话说，必须确定哪条消息将被用来传递一个给定信号。

考虑到效率，很自然地会试图将微控制器必须发送的、时间要求一致并且发送至同一组接收器的各种信号捆绑在一起，以一条或尽可能少的消息传送。

为此，有必要重新排列图 5-13 中表格的行与列，并将其扩展为一个通信矩阵。左边的列将用来放置发送信号的微控制器，信号以信号传送速率递增的顺序排列。

建立消息以后，产生如图 5-14 所示的通信矩阵。信号 S_1 由消息 N_3 传送，图 5-14 显示了若干个还没有讨论过的另外函数得到的信号和消息。

当分析实际车载网络时，必须考虑另外方面的大的偶然因素。首先，信号、消息、发送器和接收器的真实数目比本节讨论的要大得多。因为由一个给定发送器传送的信号通常由几个接收器以不同采样速率处理，所以情况会变得更加复杂。另外还值得注意的是函数分布通常受到许多附加前提的限制。

因此对需求的早期分析、实现方法的评估，以及网络设计的逐步改善显得非常重要。

一旦通信系统、消息和网络拓扑被确定，通过仿真方法，可以对总线负荷或期望的通信时延做出进一步的分析与确定。

用这种方法得到的结果包括了一个分布式和网络化系统的描述，其中所有软件函数被分配一个微控制器，通信矩阵也完全被确定。

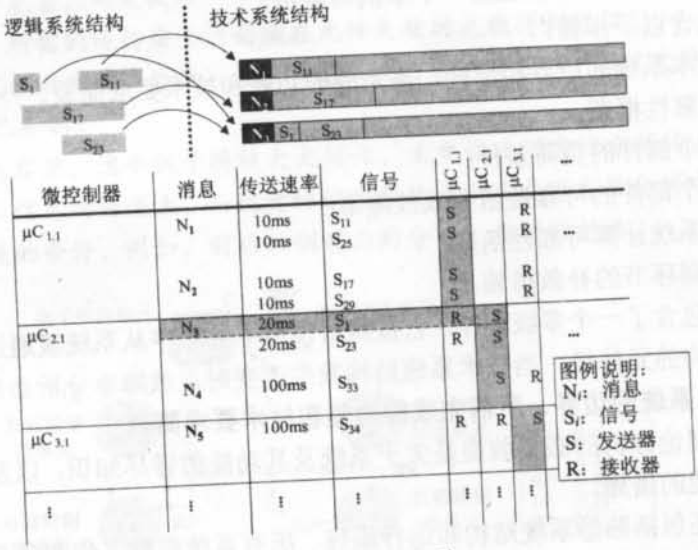


图 5-14　通信矩阵

5.2.4　可靠和安全系统的分析和描述

车辆功能的可靠性和安全性要求是客户需求和技术、法律以及财政条件相互作用的结果。例如，可靠性要求有短的修理时间或长的服务时间。相反，安全性确定了车辆在部件不正常工作或故障时的安全操作性能。从最开始，车辆函数的可靠性和安全性还确定了与技术实现和验证相关的要求。

因此，有助于可靠性和安全性分析的系统方法对软件开发的影响不断增强，例如对监控、诊断和安全性实现的影响。对于复杂的电子系统，保证可靠性和安全性的活动必须与后面的集成一起在整个项目中提早计划。

可靠性和安全性分析包含了故障率和故障模式分析，以及提高可靠性和安全性措施的研究和评估。故障模式分析也包括所谓的故障模式和作用分析（FMEA）[61]，以及故障树分析（FTA）[53, 54]。

5.2.4.1　故障率分析和可靠性函数计算

一个观测单元故障率的系统，详尽的研究有助于在计算的基础上预测它的可靠性。这个可靠性的预报，有利于尽早辨识系统的薄弱环节和评估可选择的解决方法，是非常重要的。它还有助于定量获取可靠性、安全性，和可用性相互之间的耦合关系。另外，当部件的可靠性需要以公式表示时，这方面的研究是必要的。

值得提起的是，由于故意的忽略和简化，以及不太可靠的输入数据，计算和预测出的可靠性只能用于对真正的可靠性进行估计，而真正的可靠性只能通过可靠性测试和实地试验来确定。然而，为了在分析阶段进行比较研究，绝对准确并不是很重要。因此，预测出的可靠性在评估实现方法时特别有用。

本节中，观测单元总是一个技术系统或系统部件。然而，正常状况下，更一般的观点

是可以设置更宽的范围，甚至包括一个车辆驾驶员。

故障分析包含以下步骤：

（1）定义技术系统的边界和部件、需完成的功能和技术要求简表。

（2）建立可靠性框图；

（3）确定每个部件的负荷条件；

（4）确定每个部件的可靠性函数或故障率；

（5）为整个系统计算可靠性函数；

（6）提高薄弱环节的补救措施。

故障率分析包含了一个多级程序，它按照自顶至下的顺序从系统级通过不同的子系统级到技术系统结构的元件级。当技术系统结构发生改变时，故障率分析必须重新进行。

1．定义技术系统的边界、所需完成的功能和技术要求简表

可靠性预报理论分析的必要前提是关于系统及其功能的详尽知识，以及提高可靠性和安全性的具体可能的措施。

对系统了解还包括熟悉系统结构和运行原理、所有系统部件工作和负荷条件，另外还有元件间的相互作用——可能以信号流和所有元件的输入输出的形式表示。

提高可靠性和安全性的方案包括限制或降低元件运行的静动态负载，采用具有更高适配性的元件，系统或元件设计的简化，关键元件的预处理，以及引入冗余技术。

一个所需要完成的功能表示了系统的任务。因为系统边界和所需完成的功能也会被用来定义故障，所以它们构成了任何可靠性和安全性分析的起点。

另外，因为所有系统元件的环境条件决定了元件的可靠性，所以这些必须被定义。例如，温度范围会对硬件故障率产生重要影响。就车辆本身而言，所需的温度范围，在潮湿、灰尘和腐蚀性空气中的运行情况，冲击和振动负荷，或操作电压的波动，统统属于环境条件。如果所需函数和环境条件还是时间依赖的，那么必须建立一个技术要求简表。一个由法律规定的技术要求简表的例子是用于校验驾驶期是否满足废气排放规定。在这种情况下，术语“代表性要求简表”被使用。

2．建立可靠性框图

可靠性框图表明为了满足所要求的功能，一个给定系统的哪些元件必须一直运行，以及哪些元件出故障时不会造成严重影响，因为它们是冗余的。可靠性框图通过观察技术系统构造部件来建立，这些部件被安排在一个框图中，实现功能所需的部件以串行形式进行切换，冗余部件则被并联。

举例：对一个机电制动系统建立可靠性框图

定义系统边界是建立如图 5-15 所示的假想电子制动系统的第一步。系统包括制动踏板单元（C_1）、ECU (C_2)、车轮制动单元(C_5, C_7, C_9 ,C_{11})，以及电气连接(C_3, C_4, C_6, C_8, C_{10})。

在制动传动汽缸间没有提供通常所用的液压线连接，制动踏板和车轮制动闸、机电制动系统采用电气连接。制动动作导致制动踏板单元 C_1 和车载 ECU C_2 发出的命令与制动所需的电能通过电气连接传送给车轮制动单元 C_5, C_7, C_9 和 C_{11}。这个过程有一个主要目的：在通常的制动系统中，制动踏板单元和车轮制动闸之间的“信息和能量的传递”功能是由机械和液压系统实现的。由电气和电子部件 C_2, C_3, C_4, C_6, C_8 和 C_{10} 实现的电子制动绝不会

引入附加的安全危害，而是提供一个更为安全的方式。因此，制动指令的安全性构成了主要需求。同样，所需的运行安全性必须在元件失效时也能得到保证。

下面将研究制动功能。为此，必须确定系统的总体可靠性。假设元件 C_1 到 C_{11} 的故障率 λ_1 到 λ_{11} 是已知的。

在余下的内容里，这个例子将被大大简化，主要讲述可靠性分析的主要过程。因此，研究主要集中在信息的传递上，而能量的供应和传递以及在可靠性分析中必然会被考虑的与驾驶动态有关的条件，例如，前后轴制动力的分布，都将被忽略。

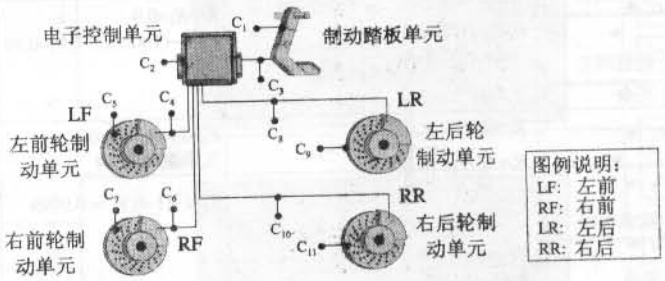


图 5-15 电子控制制动系统的结构图

在这种简化中，制动踏板单元(C_1)、ECU (C_2)以及制动踏板单元和 ECU 之间的连接，对于制动功能的实现是必需的。

车轮制动单元的功能以及 ECU 和车轮制动单元间的连接受冗余技术保护。因为简化时假设车辆的辅助制动效果仅仅使用一个车轮制动单元就可完成，所以部件 C_4 和 C_5 是必须的，而部件 C_6 和 C_7，C_8 和 C_9,和/或 C_{10} 和 C_{11} 是冗余单元。这种配置称为“一比四”冗余。基于上述观点，制动功能的可靠性框图如图 5-16 所示。

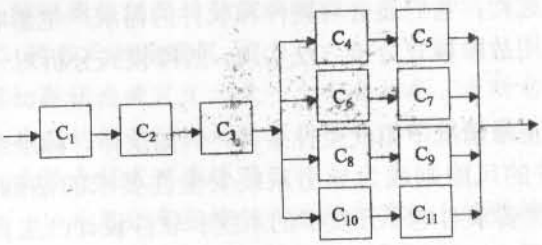


图 5-16 电子控制制动系统的制动功能的可靠性框图[57]

3. 计算系统的可靠性函数

拟定负荷条件、定义所有部件 C_i 的可靠性函数 $R_i(t)$后，根据图 5-17 所示可靠性框图的基本规则就能计算系统可靠性函数 $R_S(t)$了。

对图 5-16 中的例子来说，能够计算系统的可靠性函数 R_S。假设 $R_4 = R_6 = R_8 = R_{10}$ 且 $R_5 = R_7 = R_9 = R_{11}$，R_S 推导如下：

$$R_S = R_1 R_2 R_3 [1 - (1 - R_4 R_5)^4] \tag{5.3}$$

正如这个简化的例子表明，与元件可靠性相比，当可靠性框图中的冗余元件存在时，整个系统的可靠性会增加。相反，由串联元件构成的系统可靠性与元件本身可靠性相比是

降低的。因此应坚持这样一条道理：要求可靠性框图中串联的元件具有高度的可靠性、或引入具有冗余结构的技术系统结构是十分必要的。

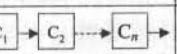
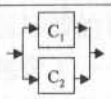
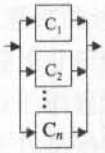
可靠性框图	可靠性函数 $R_s=R_s(t),R_i=R_i(t)$	失效率λ_s 如果λ_i为常数; $R_i(t)=e^{-\lambda_i t}$	例子
	$R_s=R_i$	$\lambda_s=\lambda_i$	
	$R_s=\prod_{i=1}^{n}R_i$	$\lambda_s=\sum_{i=1}^{n}\lambda_i$	$R_1=R_2=0.9$ $R_s=0.9\times0.9=0.18$
 1：2冗余	$R_s=1-(1-R_1)(1-R_2)$ $=R_1+R_2-R_1*R_2$		$R_1=R_2=0.9$ $R_s=1-(1-0.9)(1-0.9)=0.99$
 k：n冗余	$R_1=R_2=\ldots=R_n=R$ $R_s=\sum_{i=k}^{n}\binom{n}{i}R^i(1-R)^{n-i}$ 若$k=1$ $R_s=1-(1-R)^n$		$R_1=R_2=R_3=R_4=0.9$ 冗余度为1：4 $R_s=1-(1-0.9)^4=0.9999$

图 5-17 计算系统可靠性函数的基本规则[57]

5.2.4.2 系统安全性和可靠性分析

就安全性分析而论，一个观测单元实际上是否实现了它的功能并不重要，只要它不引入不可接受的高风险因素。增加安全性的措施所采取任何行动都称为“保护性措施”，其目的在于降低风险。

可靠性和安全性分析由若干相互关联的迭代步骤构成，就像图 4-18 所描述的那样[71]。除了对电子系统的开发过程，它们还会对硬件和软件的需求产生影响。而且，一个系统的安全性分析往往通过应用故障模式分析方法实现，故障模式分析对一个系统的所有功能进行风险评估。

所允许的风险限制正常情况下由安全性条件，例如法律、标准或规程来规定。系统功能的风险等级和所允许的风险则成为导出系统安全性要求的基础，例如根据标准 IEC 61508 [19]。系统的安全性要求对电子开发中的系统和软件设计产生重要影响。

通过故障模式分析所确定的与系统安全性相关的功能，必须引入特别的保护性措施，这些可由软硬件实现，安全性验证是准许车辆进行上路登记的必备条件。因此，在分析阶段必须计划好合适的验证和确认程序。

举例：电子节气门控制(ETC)系统监控基本原理

早在第 2 章中就确定了一个 ETC 系统的安全要求类（图 2-59）。假定一个可能的危险是无意的加速并导致车祸，对发动机 ECU 来说，这意味着所有会无意导致发动机力矩增加的控制函数f_n将被考虑与安全性挂钩。因此，需要对这些函数施加监控的措施。

在这个例子中，已经用于发动机 ECU 几年的监控原理被略微修改，它的研究将根据安全性和可靠性进行。在德国汽车工业联合会(VDA)“E-Gas”工作委员会的赞助下，由德国博世公司开发的作为汽油和柴油发动机 ECU（图 2-62）的标准化监控原理现在正得到进

一步发展。

图 5-18 表示对与安全性相关的控制功能 f_n 的监控原理。

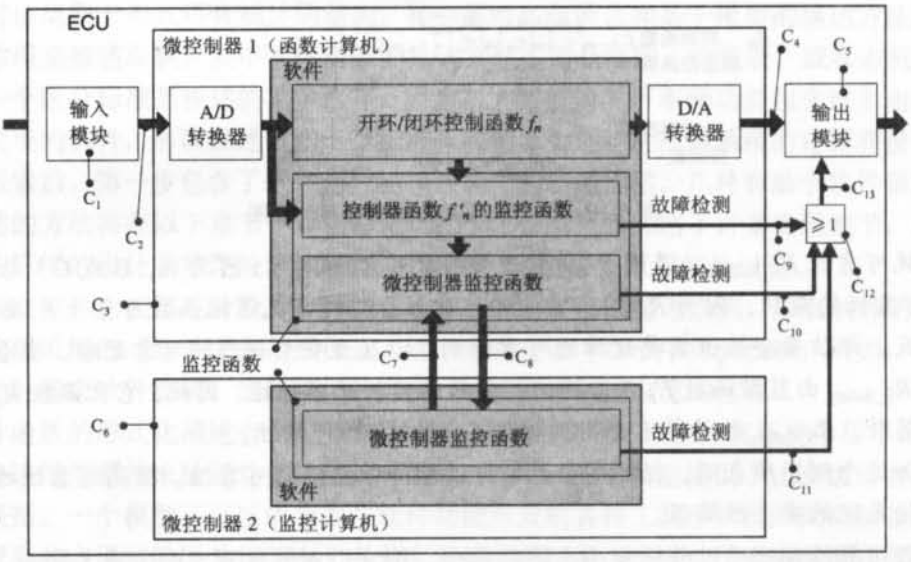


图 5-18　发动机电子控制单元与安全性相关功能的监控原理[81]

与安全性相关的控制函数 f_n 受监控函数 f'_n 的持续监控。尽管监控函数 f'_n 与控制函数 f_n 采用同样的输入变量，但它们使用不同的数据和算法工作。

除了检查 RAM，ROM 和微处理器功能，监视微控制器的监控函数还要验证控制函数 f_n 和监控函数 f'_n 是否真正在运行。这使得有必要在发动机 ECU 中使用一个辅助微处理器，即所谓的监控计算机。监视微控制器的监控函数在功能计算机和监控计算机上分布运行。两台计算机以“问-答”的形式相互监视。

机电节气门的电源切断状态被定义为是一个安全状态。当断电时，节气门自动置于怠速位置。因此，至安全状态的转移通过切断驱动节气门的控制器输出模块的电源而实现。采用这种方法，发动机在紧急模式下能连续运行。

因此不仅监控函数 f'_n 而且分布在控制和监控计算机上的监控函数也能切断节气门输出组件的电源。

如果一个故障被检测到，那么除了上面讨论的安全响应，还会有一条信息写入故障存储器中。在大多数情况下，一条合适的消息通过仪表台上的指示器传递给驾驶员。

为了评估这个监控思想的可靠性，首先必须区分三种类型的函数：

（1）控制函数 f_n；

（2）监控函数 f'_n；

（3）监控微控制器的函数。

一旦这一步完成了，这些函数的可靠性框图就能被确定了（如图 5-19 所示）：

为了确定系统的可靠性，所有这些函数在同一时刻都是需要的。相应地，系统可靠性由这些框图的串联引起。此外，部件 C_7 和 C_8 也必须串行连接。

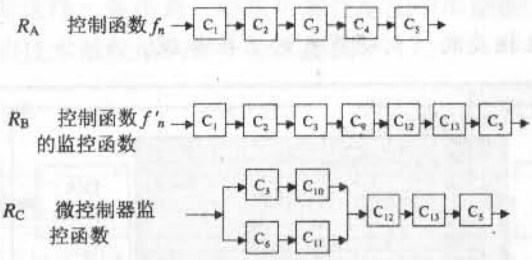


图 5-19 发动机控制功能的可靠性框图

系统可靠性 $R_{S_Reliability}$ 是图 5-19 中三个函数可靠性 R_x（x 可为 A，B 或 C）与部件 C_7 和 C_8 可靠性的乘积；因为 $R_x<1$，系统的可靠性在任何情况下比函数可靠性 R_x 要小。

相反，评估安全性仅需要故障的可靠检测，以及至安全状态的可靠迁移。安全响应的可靠性 R_{S_Safety} 由监控函数 f'_n 或监视微控制器的监控函数描述。因此，它比函数 R_x 的可靠性高。另外，R_{S_Safety} 的计算不会用到部件 C_7 和 C_8 的可靠性。

正如这个例子所表明，提高安全性的方法能降低系统的可靠性。提高可靠性的方法显然也会使系统的安全性降低。

尽管可靠性和安全性分析集中在硬件部分，但它们对软件开发产生重大影响。例如，在一个分布式网络化系统中，它们影响微控制器软件函数的分配；在软件开发中，它们决定必需的质量保证措施。有关这方面的详细讨论请参考文献[57，59，82，83]。

5.3 软件功能的描述和有效性验证

一旦确定了软件函数的接口、采样速率与软件函数至微控制器的分配，那么就可以对软件函数的数据和行为的问题进行描述，例如在算法中利用输入信号来计算输出信号。有关这方面的内容是以下各节的主题。

有几种方法可用于软件函数的描述，其中一种如图 5-20 所示。首先必须区分形式和非形式描述方法，前者采用严格的数学方法来描述算法，即算法以无误差的明确的公式化描述。相比之下，非形式方法没有那么严格。打个比方，假如形式方法采用高级语言对算法进行描述，那么用通俗语言描述算法的必定是非形式方法。

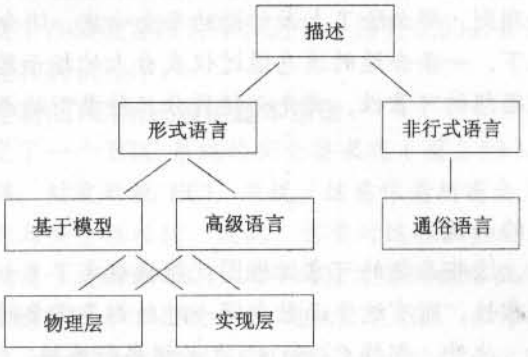


图 5-20 软件功能描述方法概要

如 4.6.2 节所讨论，简单的例子可以被用来证明算法的非形式描述固有的描述误差可以导致不同的实现。尽管输入信号相同，但这些会依次产生不同的结果和输出信号。因此，下面的讨论局限于形式语言描述的研究，其中需对高级语言和基于模型的描述方法加以区分。经常用来描述车辆开发中软件函数的描述方法实例是框图、决策表、或状态机。

另一个区分标准是描述的抽象水平。因为在大多数情况下车辆功能包含许多由不同技术方式实现的部件，所以描述需要一个标准化的抽象水准。从控制系统的建模和设计方法中获取线索后，第一步包含了软件函数物理层面上的功能描述。几种有助于软件函数基于模型描述的方法将在以下章节中详细讨论。这个讨论刚开始忽略了许多实现细节，而这些细节将在后续的设计和实现阶段被确定。这些在 5.4 节中被更详细地讨论。

如果在软件功能描述阶段，软件的要求和前提条件已经被考虑到，那么软件的描述、设计和实现阶段的一致性才能得到保证。这一点将在以下章节中得到反映，例如，在软件功能描述时考虑软件结构，实时特性的定义，或数据流和控制流的区分。

软件函数的形式化描述有利于开发。例如，描述完成后可以在仿真环境中尽早地执行，并利用快速原型系统来验证功能描述的有效性。这里，图形模型提供了比高级语言描述更简单的概括。一个模型还可以成为参与软件功能开发的各种工程学科的共同理解基础。

物理层和实现层的区分目的在于各种实现细节的抽象，这些细节有些是硬件依赖的。这样，软件功能的描述能够应用在各种不同的汽车项目中。例如，通过对软件函数到微控制器的端口采用不同字长。

5.3.1　软件体系结构和软件组件的描述

从所建立的逻辑软件函数模型及其已定义的实时性要求和输入/输出信号着手，必须确定软件函数体系结构。起始点是图 1-22 中车载 ECU 微控制器的软件结构。为了处理丰富的软件函数，一个合适的模块化和层次概念是必不可少的。因此，在软件层需要组件和接口的协调应用。描述一个给定软件函数时所使用的所有软件组件的接口定义也是这种软件系统分布式开发的主要先决条件。实现这个要求的例子是基于对象的建模，这种建模方法的好处是具有软件组件的可重用性。

5.3.1.1　基于对象的软件体系结构的建模

本节介绍与基于对象的软件模型有关的主要术语。一个软件系统被划分为所谓的对象，也就是被结构化的、互相独立的、被封装的软件组件。对象依靠完成特定的任务相互作用。除了软件行为，对象还包含软件结构[73]。

术语“结构”是指对象可以包含属性，而对象数据储存在这些属性中。换句话说，属性是对象内部的存储空间。结构还表示对象可以包含其他对象。这种对象间的包含关系称为聚合。因此，结构描述了一个对象的静态特性。

相反，术语“行为”却描述了对象的动态特性。对象访问其他对象需要接口。一个对象接口被定义为公共过程。这种过程能从对象外部获取输入数据，还能修改对象属性或提供对象输出数据。要修改一个给定对象的任何属性，都要调用那个对象的一个过程。这保证了对象的模块化、柔性和可重用性。因此，对象适用于各种环境。

一个“类”是简单对象集合的抽象，它表示每个对象的公共属性和过程。对象表示一个类的拷贝或实例。可以这么说，一个类是一个对象实例准则的描述。

例：计算防抱死制动系统（ABS）的车轮转速和车辆行驶速度

基于对象的软件模型的图形可视化使得统一建模语言（UML）获得广泛应用[84]。图5-21采用UML类框图描述了具有“Speed n”属性的“Wheel”类，这个类同时还有车轮转速的初始化过程“init_n()”，计算过程“compute_n()”，和输出过程“out_n()”。这种类型的软件组件可以用于防抱死制动系统(ABS)中。

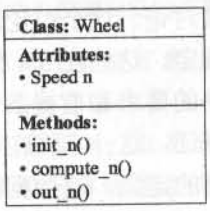


图5-21 “Wheel”类和计算车轮转速(r/min)的方法[84]

为了计算车辆速度，要定义一个“Vehicle”类。“Vehicle”类必须表示车辆的四个轮子。结果“Vehicle”类和“Wheel”类之间是一个1:4聚合。

“Vehicle”类还必须表示车辆的发动机。为此，必须对“Engine”类加以描述。“Engine”类中，属性“Speed n”描述发动机速度，另外还定义了初始化过程“int_n()”，计算过程“compute_n()”，以及发动机速度的输出过程“out_n()”。

图5-22表示了“Vehicle”类的UML语言模型。“Vehicle”类中，定义了属性“Velocity v”和“Gear g”，还有车辆速度和齿轮选择的计算过程“compute_v()”和“compute_g()”。这些计算利用了“Wheel”类和“Engine”类的过程。

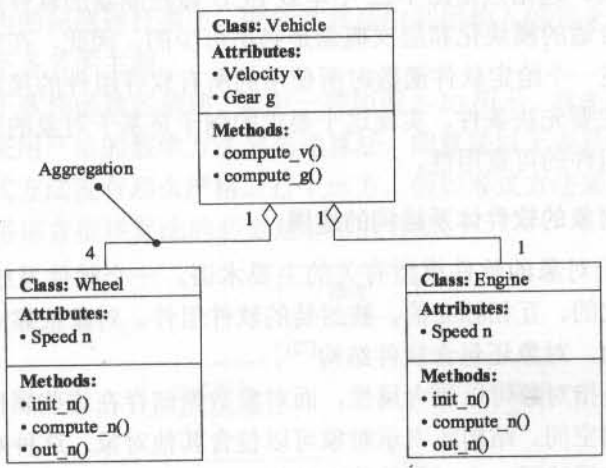


图5-22 “Vehicle”类的类框图[84]

这种软件模型被用来描述软件函数内相互关系的基础。例如，一个软件函数f1通过这种方法可由软件组件层次化地构成。这是由类的实例化实现的。这样，“Vehicle”类的实例化建立了名为“Vehicle_1”的对象。这个对象包含了4个“Wheel”类实例，和一个“Engine”

类实例。所建立的对象是的“RF_Wheel”，“LF_Wheel”，“RR_Wheel”，和“LR_Wheel”，它们分别对应右前轮，左前轮，右后轮，和左后轮，外加发动机的对象“Engine_1”。

以下章节中，包含对象的软件结构表示采用类似图 5-23 的图形可视化方法。一个对象的类的名称在对象名称后标出，并用“:”（冒号）分隔。

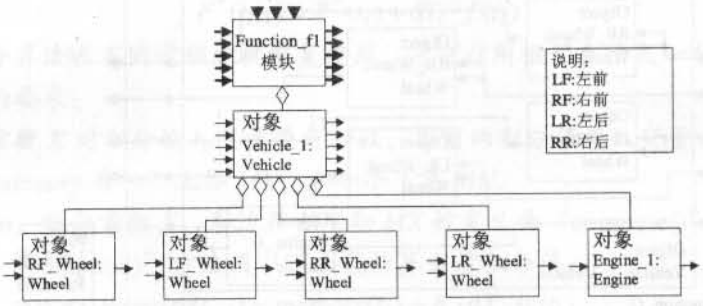


图 5-23　包含对象框图的软件结构图形描述

5.3.1.2　与实时操作系统接口的模块化描述

软件功能中处于最高层次上的软件组件要求与实时操作系统的接口以过程的形式定义，与通信系统的接口以消息的形式定义（如图 2-36 所示）。把这些过程分配给任务有助于实时性要求的描述。通过以消息的形式复制软件函数的输入输出信号，支持任务间甚至是微处理器间的通信。

除软件函数中顶层外的其他层并不要求使用这种接口。因此，软件函数顶层的软件组件称为“模块”。在接下来的讨论中，模块以图形的形式如图 5-24 中表示。过程 P_1 到 P_m 以三角形表示。消息 M_1 到 M_n 以箭头表示，依靠箭头的方向可以分辨出接收和发送的信息。

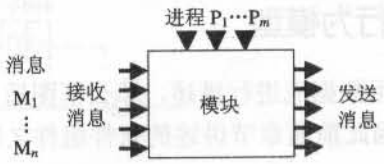


图 5-24　模块的图形表示

5.3.1.3　基于类的可重用软件组件的描述

在更低的层次上，所定义的对象可以被过程通过方法访问。一个方法的接口可以包含若干个参数和一个返回值。模块和对象在软件组件中的区分有利于对象在各种语境中重用。在下面的讨论中，类和对象如图 5-25[73]中的图形描述。变量用箭头表示指向类或对象。返回值用从类或对象发出并离开的箭头表示。

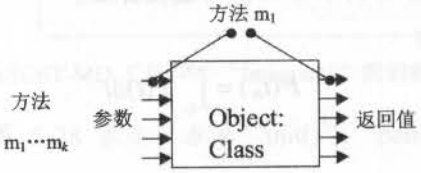


图 5-25　类和对象的图形表示

作为图 5-23 函数结构可视化的替代，下面用图 5-26 所示的框图进行讨论。除了表示集合和层次关系，这个图还提供了对象过程、参数和/或返回值之间的数据流和控制流。这些路径以线条表示，如图 5-26 所示。

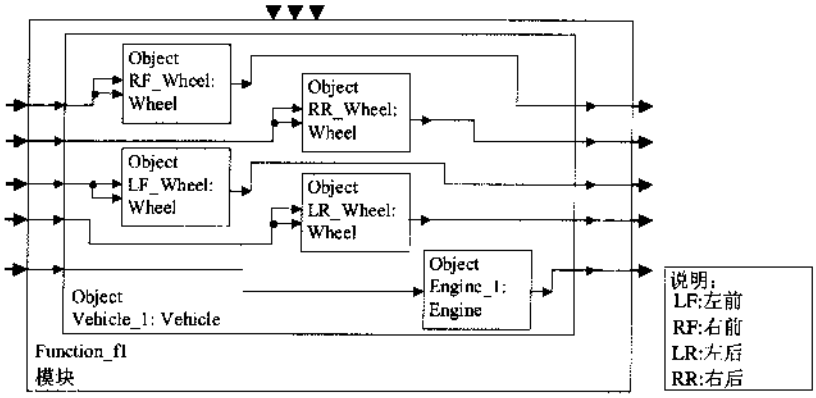


图 5-26 包含框图的软件结构的图形化描述[73]

依赖于所需描述的语境类型，有若干种不同的建模技术可供选择用来描述模块和类的行为。其中，框图、决策表和状态机是最重要的技术。

5.3.2 数据模型描述

对软件组件数据模型的描述在抽象层进行。在基于对象的建模中，数据由对象和/或模块的属性提供。

5.3.3 用框图描述行为模型

如果软件组件行为能够用数据流进行描述，那么框图适合用来做图形可视化描述。这在抽象层上是经常发生的。因此前面章节讲述的软件组件之间的众多内部联系可以用框图描述。

框图能使软件组件中的复杂算法得到清晰的结构可视化，并能对数值函数和布尔函数进行区分。

5.3.3.1 数值函数描述

作为描述数值计算的例子，这里用框图描述欧拉积分算法（图 4-36）。

举例：“积分类”的描述

欧拉积分算法通常被用来作为积分计算的逼近算法。

函数 $f(t)$的定积分计算 ：

$$F(t_n)=\int_{t_0}^{t_n} f(t)\mathrm{d}t \tag{5.4}$$

可由下面的求和计算近似：

$$F^*(t_n)=\sum_{i=0}^{n-1}(t_{i+1}-t_i)\cdot f(t_i) \tag{5.5}$$

$(t_{i+1}-t_i)$称为步长 dT_i。

$F^*(t_{i+1})$可以由下式递推计算：

$$F^*(t_{i+1})=F^*(t_i)+dT_i\cdot f(t_i) \tag{5.6}$$

因为这个积分算法在不同过程中被重复使用，它可以用框图表示成一个类。这对软件组件提出了额外的要求：

（1）用一个常数 *K* 对积分输入变量进行加权，当前的积分值存入变量 *memory* 中。

（2）积分值 *memory* 有一个上界 *MX*，和一个下界 *MN*。

（3）积分值 *in*，积分常数 *K*，和边界 *MN* 和 *MX* 被定义为“compute()”方法的缺省参数。在此基础上，“compute()”方法用以下公式计算当前积分值：

$$memory(t_{i+1})=memory(t_i)+K\cdot dT\cdot in(t_i) \tag{5.7}$$

并用边界 *MN* 和 *MX* 限制积分值的范围。

（4）积分步长是最后一次执行“compute()”方法以来所过去的时间；同样的值在图 2-18 中称为执行速率。实时操作系统为每个任务计算并提供步长 dT。

（5）使用方法“out()”输出当前积分值 *memory*。

（6）“init()”方法用值 *IV* 对 *memory* 进行初始化，这个值以参数的形式被传递。

（7）方法“init()”或“compute()”的执行依赖于布尔参数 *I* 或 *E*，这两个参数在每一种情况中以附加参数的形式被传递给这些方法。

ASCET-MD 工具[73]中框图形式的积分器可视化如图 5-27 所示。数值和布尔数据流分别用实线箭头和虚线箭头表示，控制流则用点划线箭头表示。

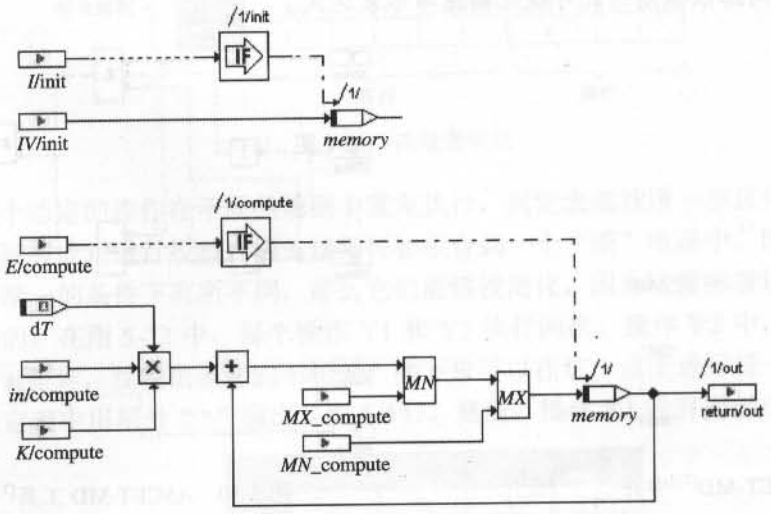


图 5-27　ASCET-MD 工具[73]中“Integrator”类的框图描述

软件组件的外部视图如图 5-28 表示。方法“init()”、“compute()”和“out()”的参数分配如图 5-27 所示。

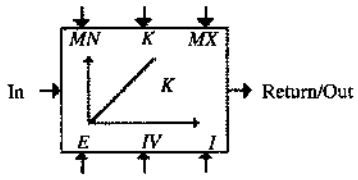


图 5-28 ASCET-MD 工具[73]中“Integrator”类的外部视图

值得注意的是数值算法是在逻辑层被定义的。这样，上述例子没有反映与后续执行有关的任何定义，例如字长的定义，或与定点或浮点运算有关的定义。因此，采用这种方式描述的软件组件适用于各种微控制器。另一个发生在逻辑层的步骤是限制积分值。它在浮点运算的实现过程中执行，并且不能与整数运算的溢出处理中的限幅相混淆。然而，算法操作或控制流的顺序序列已经被定义了。

5.3.3.2 布尔函数描述

除了数值运算的描述，框图也被用来定义逻辑运算，也就是所谓的布尔运算。在许多情况中，即当依赖于布尔参数的操作采用“IF ... THEN”形式描述时，需要用布尔运算和数值运算描述一个函数。

假设布尔变量的取值范围为集合 B = {TRUE, FALSE}的两个元素值。布尔变量与布尔运算符连在一起构成布尔表达式。对于布尔运算符与（逻辑 AND 运算），或（逻辑 OR 运算）以及非（逻辑 NOT 运算）、以下章节采用图形符号——也就是开关函数（如图 5-29 所示）。

举例：布尔表达式的框图描述

使用开关函数的符号描述布尔表达式称为切换网络。图 5-30 所示为 ASCET-MD 工具[73]中用一个切换网络以框图形式可视化描述布尔表达式。

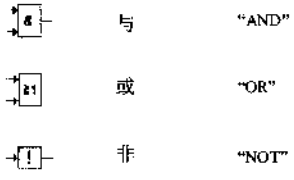


图 5-29 ASCET-MD[73]中开关函数的图形符号

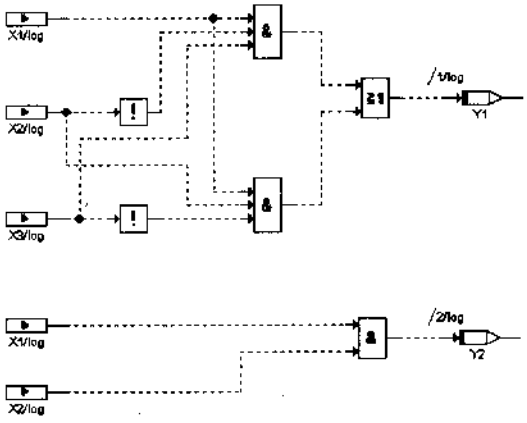


图 5-30 ASCET-MD 工具[73]中的布尔表达式的框图描述

5.3.4 基于真值表的行为模型描述

作为一种选择，任何依赖于条件的执行可以通过所谓的判定表进行简洁而结构清晰的

定义[73, 74]。一张判定表的输入和输出变量分别包含布尔变量 X1~X*n* 和/或 Y1~Y*n*。

X1~X*n* 中的每个输入变量，也称为条件在判定表中表示为列。这样，判定表中输入变量的每一行表示各个输入变量的与运算，或逻辑 AND 函数。因此，一行包括一个布尔表达式，即所谓的规则 R，它的真值确定了输出变量 Y1~Y*m*，也就是操作。这些输出变量作为判定表的附加列。

n 个输入变量最多有 2^n 种组合。因此完整的判定表包括 2^n 行或规则。输出值被分配给一条或更多的规则。如果一个输出变量被分配若干条规则，那么这表示对应规则的“或”即逻辑 OR 操作。图 5-31 所示为图 5-30 中布尔表达式的真值表。这样，图 5-31 和图 5-30 是等价的。

	X1	X2	X3	Y1	Y2
R1	0	0	0	0	0
R2	0	0	1	0	0
R3	0	1	0	0	0
R4	0	1	1	0	0
R5	1	0	0	0	0
R6	1	0	1	1	0
R7	1	1	0	1	1
R8	1	1	1	0	1

相关规则　　条件　　操作

图 5-31　布尔指令的真值表描述

真值表可用与布尔表达式相同的方法进行优化。因此，图 5-31 中，仅有最后三条规则 R6~R8 是相关的，因为仅仅只有这些规则决定操作 Y1 或 Y2 中谁将被执行（图 5-32）。

	X1	X2	X3	Y1	Y2
R6	1	0	1	1	0
R7	1	1	0	1	1
R8	1	1	1	0	1

相关规则　　条件　　操作

图 5-32　决策表优化

如果一个给定的操作在不同的规则中重复执行，判定表能被进一步优化。一开始，这个操作的规则被成对进行校验。两条规则可被组合到一个“或”电路中。因此，如果这两条规则仅在唯一的条件下有所不同，那么它们能够被简化，因为这意味着这个不一致的条件是不相关的。在图 5-32 中，每个操作 Y1 和 Y2 执行两次。操作 Y2 中，两条规则仅在条件 X3 处有差异，反映出条件的不相关。接下来可以在这一点上进行进一步简化。不相关条件在判定表中用星号“*”标出（图 5-53）。然而，操作 Y1 没有提供进一步优化。

	X1	X2	X3	Y2
R6	1	0	1	0
R7	1	1	*	1
R8	1	1	*	1

相关规则　　条件　　操作

图 5-33　对操作 Y2 进行真值表优化

若干判定表也可以按一定顺序链接，如图 5-34 所示。判定表适合于函数的描述，这些函数中的组合条件导致不同操作的执行。这种相互影响在监控函数中频繁发生。有关判定表的详细讨论，请参考文献[72, 73]。

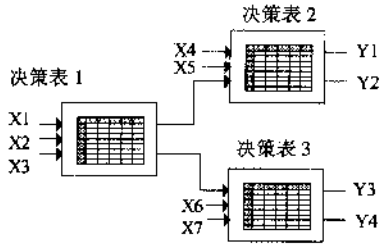


图 5-34 决策表的顺序链接

5.3.5 行为模型的状态机描述

在许多软件函数中，运行结果不仅依赖于输入而且依赖于某个事件及事件发生前的历史。这种相互作用适合于用状态机描述。本节所讨论的状态机主要是有限状态机，如 Moore、Mealy、和 Harel[72, 73, 85]。

状态机可以绘制成状态图，其中状态用带标志的圆角矩形框表示。有效迁移用带文本标志的箭头表示。迁移的发生依赖于迁移的条件。

5.3.5.1 平面状态机

作为软件函数状态自动机描述的例子，下面将继续讲述低油位指示灯的控制（图 2-9）。

举例：低油位指示灯控制的状态机描述

与低油位指示灯的控制相关的仅仅是条件“Signal value > 8.5V”和“Signal value < 8.0 V”，以及当前的状态“灯灭”或“灯亮”（图 5-35）。

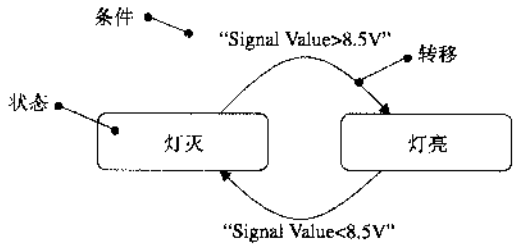


图 5-35 状态、转移和条件的描述

（1）至此，操作“开灯”和“关灯”的执行时刻还没有被确定。正像那些带有附加条件的情况，这些操作可被分配给迁移，并被称为迁移操作。这种状态机称为 Mealy 状态机。作为一种选择，操作可被分配给状态，称为状态操作。这种状态机称为 Moore 状态机。Mealy 和 Moore 状态机也可以组合在一起，即操作可被分配给状态和迁移。本例中，操作“开灯”和“关灯”将被分配给迁移。

（2）另外，状态机在起始时刻的状态需要定义。这个状态称为初始状态。为了监控低油位指示灯的运行，要定义出每当汽车发动机起动时灯的通电情况。这样，每次起动时，

可以判断指示灯是否正常工作，而不依赖于实际的油位。第一个可能的状态迁移仅在一个2s 的延迟后发生，即在起动发动机后灯将至少亮 2s。因此，新的状态“功能检验”与初始状态“灯亮”一起被引入。这样，操作“开灯”在“功能检验”状态被执行。状态机的初始状态用“(S)”标志。

通过上述步骤建立的状态机如图 5-36 所示。

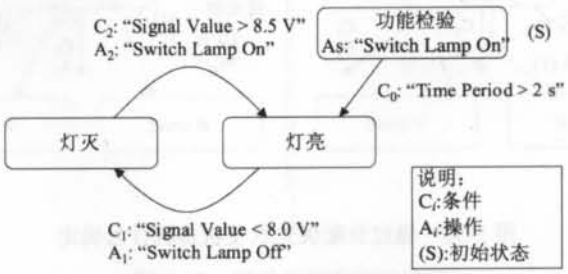


图 5-36　操作的分配和初始状态的定义

分配操作给状态时需对以下内容进行区分：

（1）仅在进入一个状态时执行的操作（进入操作）；

（2）在退出一个状态时执行的操作（退出操作）；

（3）停留在一个状态时执行的操作（静态操作）。

图 5-37 表明一个进入操作与一个迁移操作等效，这个迁移操作是分配给导向一个给定状态的所有迁移的。类似地，一个状态的退出操作也与一个迁移操作等效，这个迁移操作是分配给导离一个给定状态的所有迁移的。

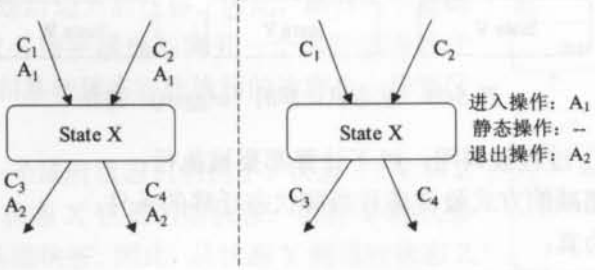


图 5-37　状态机的等效操作

如果从每个状态出发，对于每个输入，至多仅有一个状态迁移，那么状态机的行为称为是确定的。

非确定的情况是，如离开某个状态的不同迁移的若干条件在同一时刻为真。这种情况可以通过分配优先权的方法排除，即给离开某个状态的迁移分配不同的优先权。优先级正常情况下用数字表示。在我们的例子中，值较大的数字表示较高的优先级。

在图 5-38 中，有三个迁移离开状态 X。如果条件 C_2 为真，那么图左半部的状态机行为是不确定的，因为两种迁移都是有可能的。引入优先级，如图右半部，迁移与操作 A_2 执行的优先级一起定义为 3。

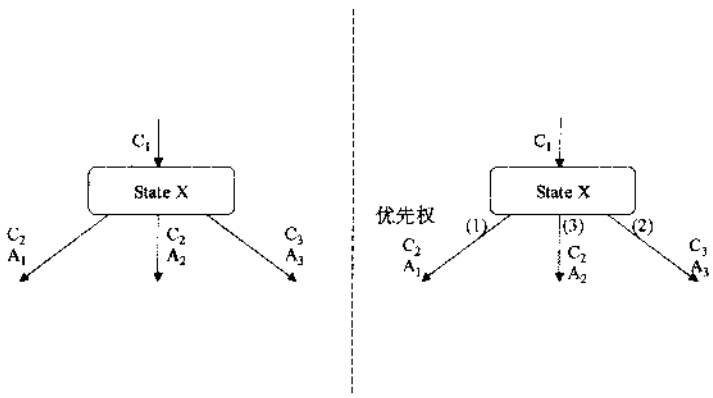


图 5-38 通过分配优先权使状态机行为确定

一个状态机还需要定义“事件”，也就是离开当前状态的迁移条件的证实，如果“事件”发生，相应的操作和迁移被执行。例如，图 5-39 中，触发状态机计算的“事件”由分配给每个迁移的“trigger()”过程描述。

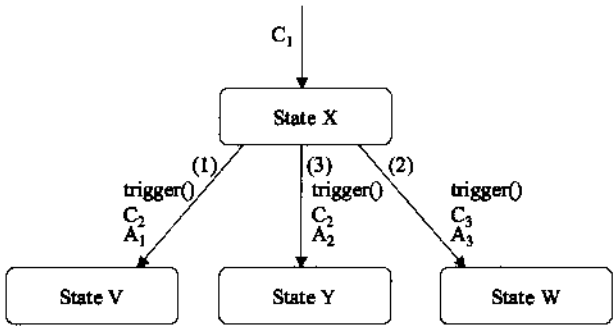


图 5-39 状态机计算的“trigger()”方法

每次“trigger()”过程被调用，以下计算都要被执行：

（1）以优先级递减的方式验证离开当前状态迁移的条件；

（2）如果条件为真；

（3）当前状态退出操作的执行；

（4）迁移操作的执行；

（5）新状态进入操作的执行；

（6）至新状态的迁移；

（7）如果没有条件为真；

（8）当前状态静态操作的执行。

5.3.5.2 用转移指令描述迁移

如果形如 C_1 & C_2，或 C_1 & C_3 的条件发生在离开同一个状态的迁移上，那么采用分支迁移可以得到更清晰的结构化。图 5-40 中的两个图是等效的。

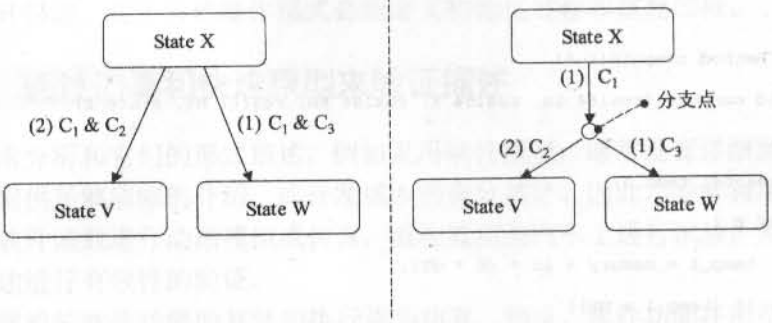


图 5-40　等效的状态迁移模型

5.3.5.3　递阶状态机

随着状态和转移数量的增加，状态图会变得复杂和混乱。然而，通过使用递阶嵌套状态可以保持状态图的清晰。

（1）对每个递阶状态，一个基础状态被定义为初始状态。那些到递阶状态的迁移，引起在初始状态的一个迁移。

（2）作为一种选择，可以定义一个带记忆的递阶状态。用“H”标识一个递阶状态的历史“History”，到这个递阶状态的每个迁移导致这个状态的最后有效基础状态被再一次呈现。这样，初始状态定义了首次进入递阶状态的基础状态。

还可以定义跨过递阶边界的迁移。因此，离开一个基础状态的迁移的优先权必须明确地与离开一个递阶状态的迁移区分开。递阶状态和基础状态操作执行的次序也一样要区分开。

图 5-41 所示为一个递阶状态机的例子。状态 X、Y 和 Z 在最高递阶层定义。状态 X 也是初始状态。状态 V 和状态 W 是递阶状态 Z 的基础状态。因此，从状态 Y 到递阶状态 Z 的迁移导致进入初始状态 V 的一个迁移，这与跨过状态 Z 的递阶边界从状态 X 到状态 V 的直接迁移是一样的。

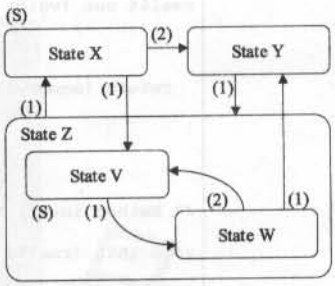


图 5-41　递阶状态机

5.3.6　行为模型的高级语言描述

在许多情况中，高级语言是定义软件组件行为的较好方法。它特别适用于困难或不清晰的面向数据流或基于状态的描述。由于有众多的循环和分支，搜索和排序算法就属于这种情况。

举例：“积分器”软件组件的 C 语言描述

图 5-42 所示为图 5-27 中“积分器”软件组件 C 语言描述的方法[76]。

```
/* Variables */
extern real64 memory;
extern real64 dT;

/* Method compute() */
void compute (real64 in, real64 K, real64 MN, real64 MX, sint8 E)
{
  real64  temp_1;
  if E {
    temp_1 = memory + in * (K * dT);
    if (temp_1 > MX){
      temp_1  = MX;
    }
    if (temp_1 < MN){
      temp_1  = MN;
    }
    memory = temp_1;
  }
}

/* Method out() */
real64 out (void)
{
  return (memory);
}

/* Method init() */
void init (real64 IV, sint8 I)
{
  if I{
    memory = IV;
  }
}
```

图 5-42 “Integrator”类的 C 语言方法描述[76]

5.3.7 实时模型的描述

除了软件组件的数据模型和行为模型描述外，实时模型描述也是需要的（图 4-31 和

4-32)。如果使用一个实时操作系统，必须定义它的配置。除了各种操作模式、迁移和迁移条件，还必须定义判优策略的描述以及每个操作模式的任务过程表。

通过模块的过程和消息接口，以及通过步长 dT 计算，使得模块和类的动态的实时性要求可以分开描述。对于各种操作模式必须定义初始化过程和重复过程。

5.3.8　通过仿真和快速原型来验证描述

软件需求分析和它们的形式描述，例如采用软件模型，通常没有详细到能给一个被开发软件系统提供足够清晰的介绍，或开发成本的充分估计。因此，经常利用方法和工具对形式描述的软件函数进行动画模拟或仿真，或在真实的汽车上进行试验，为的是尽早对软件功能的描述进行有效性的验证。

基于计算机的软件功能的复现和执行称为仿真。相反，软件功能在所谓实验系统上的执行，即一台计算机通过接口与车辆互联，称为快速原型。

如果软件模型用来作为仿真和快速原型方法的基础，那么模型编译器是需要的。这有助于将模型描述直接或间接地编译成适于仿真或实验系统执行的机器代码。在这个过程中，模型编译器所要求的设计决策要么在模型中隐含定义，要么由模型编译器初始化定义，采用的方法必须保证被描述模型得到最精确的复现。

图 5-43 所示为快速原型系统工具的组成结构[73]。首先要用模型编译器把由模型工具定义的软件函数模型编译成源代码。第二步，编译工具将这些源代码编译成实验系统的程序版本和数据版本。程序和数据版本通过下载工具或 Flash 编程工具下载到实验系统中，显示它们“准备运行”。后面的程序执行能通过实验工具进行控制、参数化、动画模拟和观测。

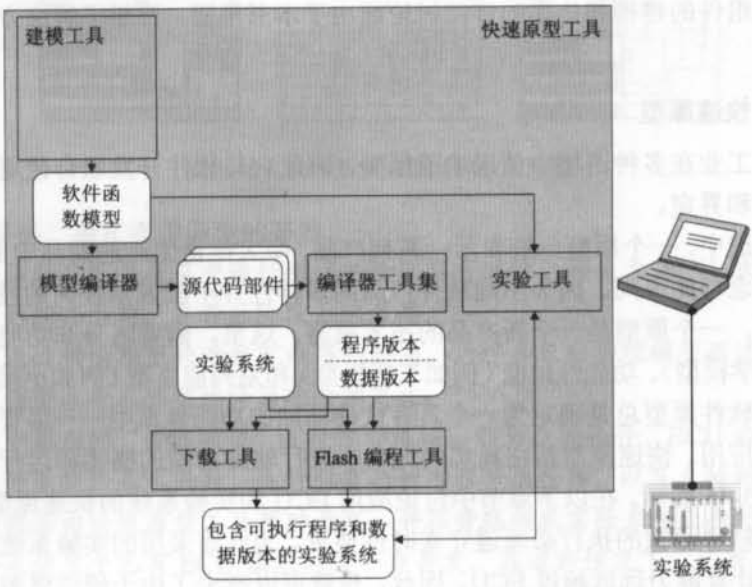


图 5-43　快速原型工具的构成[73]

在这种情况下，软件模型除了表示设计和实现的基础外，还提供建立仿真和快速原型方法的基础。一个实验系统的使用使软件函数的有效性能尽早得到验证，而且这验证是独立于 ECU 的。还有另一个好处是，实验系统随后还能作为 ECU 正确性验证的参考。

5.3.8.1 仿真

在许多情况中，仿真的目的不仅仅是软件功能的复现；更重要的在于软件函数和硬件、设定点发生器、传感器和执行器以及被控对象之间的相互作用。

从软件开发的角度看，这意味着建模还必须考虑这些因素，这些因素称为环境部件。使结果，虚拟车辆、驾驶员和环境模型被链接到虚拟的 ECU 模型和软件模型上。然后，这个模型可在仿真系统上运行，例如，在一台 PC 机上（如图 5-44 所示）。这个方法也适合于非软件实现的车辆功能的仿真。

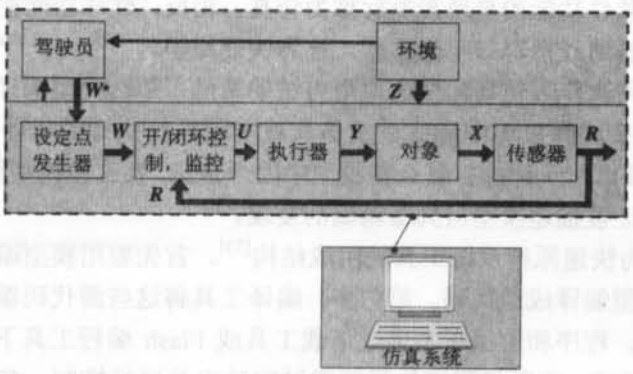


图 5-44 软件函数和环境部件的建模和仿真

有关环境组件的建模和仿真的详细讨论超出了本书范围，要想了解这方面的内容请参考文献[35]。

5.3.8.2 快速原型

因为汽车工业在多种语境中使用术语原型，因此它与软件开发联合使用时有必要给出更准确的定义和界定。

在车辆制造中，一个原型一般表示一系列产品，即大批量生产的商品中的第一个样品。软件原型的概念与此不同。因为不同软件产品的多样性并不能表示任何的技术问题。

一般说来，一个原型是一个新产品的技术模型。这里，需要区分非功能原型（例如风洞的空气动力学模型）、功能性原型（例如车辆原型）和系列前原型（例如小批量系列车辆）。

本书中，软件原型总是确定为一个具有软件功能的功能性原型——尽管实际使用中有着不同目的和应用。快速原型指在真实的汽车中进行软件功能的描述和运行的一系列方法的统称，如图 5-45 所示。在以下章节中讨论应用 ECU 和实验系统的快速原型的多种方法。

实验系统软件函数的执行必须遵守实时性要求。为此而采用的实验系统主要是实时计算机系统，其计算能力远远超过 ECU。因此，也就可以省去了由于硬件资源限制所需的软件优化。在这种给定条件下，模型编译器能够尽可能准确地对模型所描述的行为进行编译。

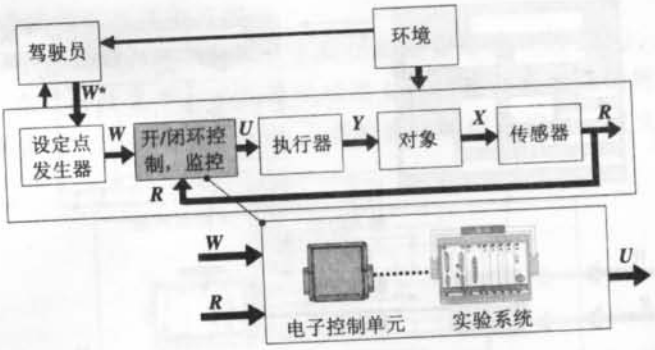


图 5-45 在实际车辆中软件功能的快速原型

可以构造模块化实验系统以适应具体的应用，例如，为输入和输出信号提供所需接口。整个系统被设计为面向车载应用，通过一台 PC 机进行操作。因此，软件函数描述的有效性直接在汽车中得到验证，或者修改软件功能的描述，同时也能够对程序和数据描述进行改进。

5.3.8.3 横向和纵向原型

原型开发要区分两个程序目标：

（1）横向原型的目的在于软件系统宽范围的描述。然而，它们仅提供了抽象描述，忽略了细节。

（2）纵向原型为受限范围的软件系统提供了非常详细的描述。

利用一个软件系统的片断，图 5-46 描述了横向和纵向原型的程序目标。



图 5-46 软件系统的横向和纵向原型

举例："旁路"模式横向原型的开发

新的软件函数应尽早在车中进行测试并验证其有效性。而关于 ECU 软件系统的实现细节，例如它的软件结构，在开始阶段无须考虑。

这个问题通过横向原型的开发可以得到解决。软件函数由物理模型描述。原型实现的许多方面通过模型本身或模型编译器隐含预定义。

函数开发中采用的"旁路模型"支持这种方法。作为先决条件，ECU 必须提供经过有效性验证的软件系统的基本函数，运行所有的传感器和执行器，以及支持到实验系统的旁路接口。函数概念由快速原型工具帮助开发，并在旁路模式实验系统上执行（图 5-47）。

这个方法也适用于已有 ECU 函数的进一步开发。在这种情况下，已有函数仍在 ECU 中计算，但按照以下方式进行修正：输入值通过旁路接口进行传递，并使用新开发的旁路函数的输出值。

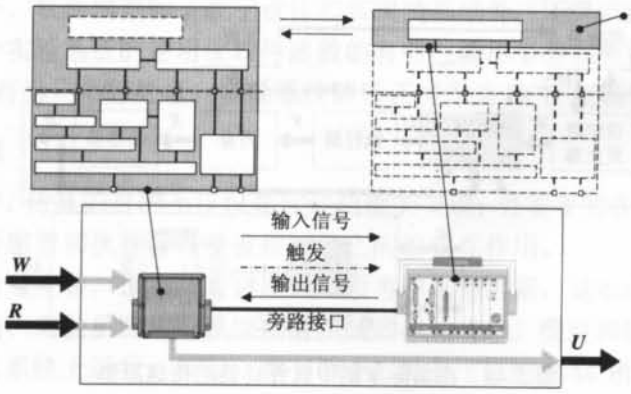


图 5-47 使用旁路系统的原型开发

在许多情况下，旁路函数的计算由 ECU 通过一个控制流接口触发（图 5-47 中的“触发”）。ECU 监控旁路函数输出值的合理性。在这种情况下，ECU 和实验系统同步运行。作为一种选择，可以实现无触发的异步通信。

在与安全性有关的函数进行工作时，当 ECU 接收到实验系统不合理的输出值时，例如，当旁路函数提供非法输出值，或实验系统和 ECU 间的通信可能会失败，或旁路函数的计算时间超量，ECU 能通过自动切换到已有内部函数或作为备用的替代值。

这种监控概念的使用能够保证在现场试验和试驾时，即使实验系统失败也仅会导致有限的人身伤害或任何车辆元件有限的损坏。这表明，旁路模式的函数开发也能用于安全相关函数的有效性验证。

至于旁路通信，应该记得 ECU 内的软件函数通常被分为若干过程，这些过程在各自的任务中被计算。在这种情况下，旁路通信必须支持各种任务对应的采样速率，图 5-48 所示为 ECU 和实验系统间给定采样速率的一个典型旁路通信序列。

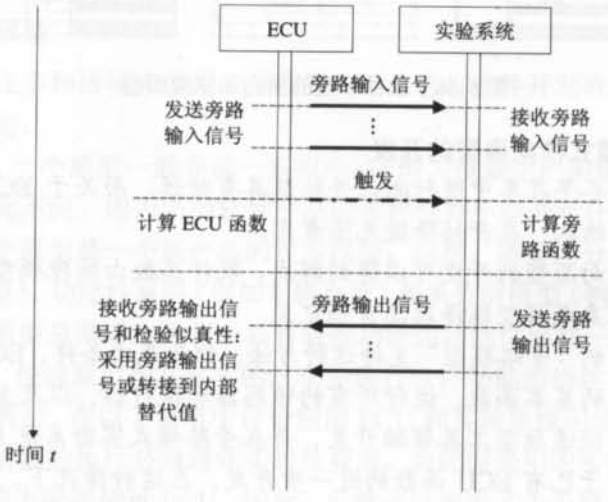


图 5-48 ECU 和实验系统间的通信

举例："全通"模式纵向原型的开发

如果必须开发一个新函数，或者如果没有提供配备旁路接口的 ECU，那么实验系统可用来开发纵向原型。在这种情况下，实验系统必须支持对应函数需要的所有传感器和执行器接口。另外还需要实时行为的定义，必须由实验系统保证遵守这一定义（图 5-49）。

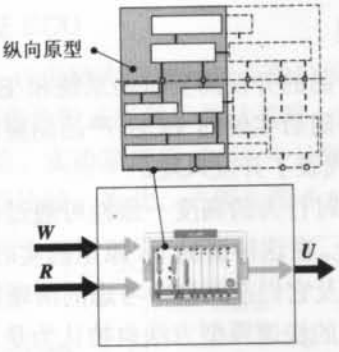


图 5-49　采用全通系统的原型开发

旁路应用适用于只有一些软件函数需要开发，且有一个配备了经有效性验证的软件函数的 ECU，例如，可能来自以前的开发项目。这个 ECU 需要修正以便能支持旁路接口。旁路应用也适合用于在一个 ECU 中具有大量传感器和执行器系统，且通过应用实验系统可能需要巨大花费的情况，例如汽车发动机 ECU。

如果这样的 ECU 不存在，附加的传感器和执行器需进行有效性验证且传感器和执行器数目有限，那么全通应用通常是较好的选择。在这种情况下，实时行为必须得到实验系统全通计算机的保证，而且可以要求额外的监控。因此，一个实时操作系统正常情况下都运行于全通计算机上。

因为旁路和全通系统的混合形式提供了高度的柔性，所以经常被采用（图 5-50）。这有助于附加传感器和执行器的集成。在一个如此装备的实验系统上，新的软件函数可以被测试，然后与 ECU 已有的软件函数一起被执行。

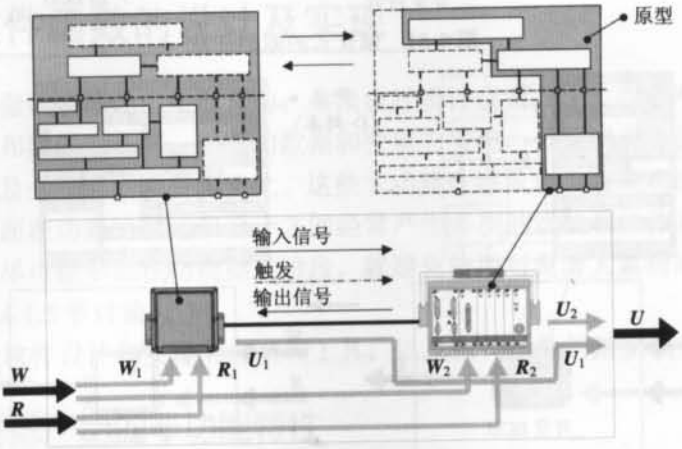


图 5-50　采用实验系统的原型开发

与 ECU 相比，一个实验系统提供了更强大的计算能力。因此，在 ECU 上实现一个软件函数需要考虑的许多要求（例如定点计算或有限硬件资源）可以安全地被忽略。这样，软件函数的修正变得更简单、更迅速。此外，实验系统中附加 I/O 接口的集成有助于各种实现方案的早期评估，如传感器和执行器。

5.3.8.4 目标系统一致原型

在大多数实例中，系列化产品的开发需要试验系统和 ECU 行为的高度一致性。仅在这种情况下，应用原型的效果与随后实现的 ECU 产品期望的行为保持一致。试验系统和作为目标系统的 ECU 间的差别代表了开发风险。

例如，试验系统和 ECU 实时行为的高度一致性可通过使用实时系统获得，例如，在两个平台上根据 OSEK 进行试验。在这种情况下，原型的实时行为称为是目标系统一致的。此外，在以后的 ECU 中实现以及它们在原型中考虑的清晰描述的设计决策，对降低开发风险是有益的，显示出这一方向的快速原型方法也被认为是目标系统一致原型。

5.3.8.5 废弃和进化原型

另一个区别的准则是，一个原型是否将被用来作为产品开发的基础。如果回答是肯定的，那么原型称为是进化的。如果回答是否定的，那么标记为废弃原型。例如，一个用于函数或工程描述的原型因为其结果不能成为产品的一部分而被确认为是废弃原型，这两种方法在汽车工业中被广泛使用，开发中进化方法最著名的例子是 ECU 软硬件的 A、B、C 和 D 样本的进化（图 5-51、图 5-52）。

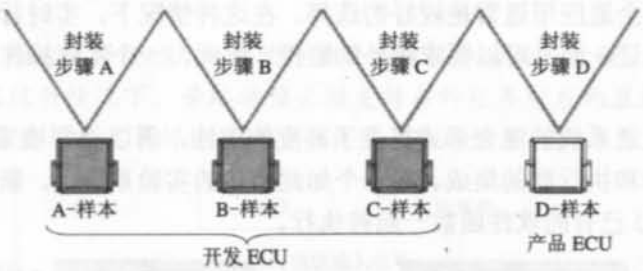


图 5-51 演化式的原型开发

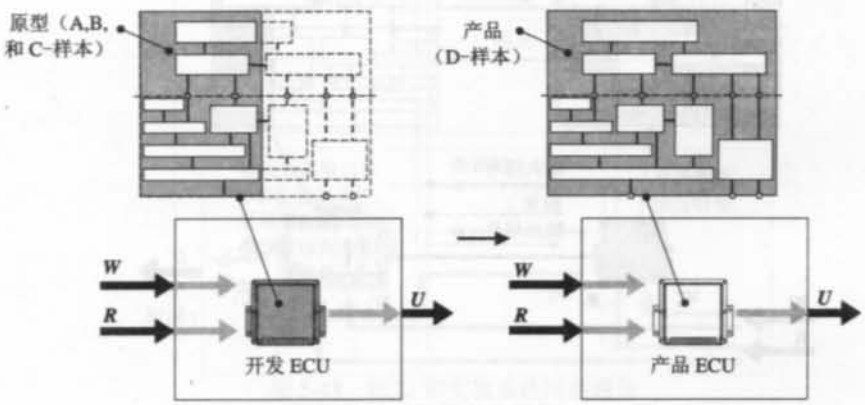


图 5-52 演化式的系统开发

应用实验系统的同时，越来越多的开发步骤移到开发 ECU 中来，通过有效性验证的系统功能描述为考虑微控制器每个细节的设计和实现提供了基础。

开发步骤转换到 ECU 正日益增多，因为利用代码生成技术，能为实验系统和 ECU 直接从软件模型生成源代码，在这个过程中还可以考虑许多的实现细节。

5.3.8.6　用参考原型验证 ECU

在目标系统一致原型中，以旁路方法进行过有效性验证的软件函数，为通过自动产生代码而实现的 ECU 函数的有效性验证提供了测试基准。为了完成这个任务，ECU 和实验系统并行且同步计算软件函数。实验系统将 ECU 中计算的软件函数的中间和输出变量与通过旁路函数获得的结果进行比较，为此，必须如图 5-53 那样扩展旁路通信。

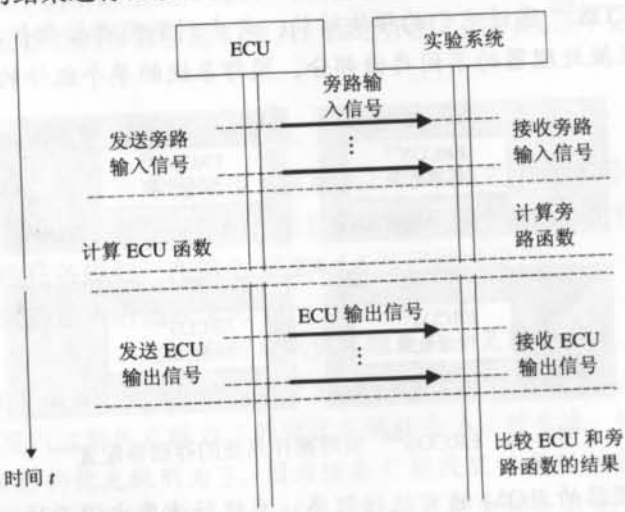


图 5-53　通过与实验系统旁路函数的比较验证 ECU 函数

在这个过程中，通过在实验系统上应用覆盖分析工具可以获得很高的测试覆盖率。

5.4　软件函数的设计及实现

软件函数在微控制器中的实现之前，必须做好设计决策。这也涉及考虑产品的非功能性要求，如程序和数据的分离，程序的数据和变量的实现，必要的非车载口支持，浮点数的算法实现，以及必要的硬件资源优化。这些非功能性要求对于系统功能的描述也会产生一定的影响，因而在功能性描述和设计之间经常产生多次的迭代和协作是很必要的。其中的一个例子是，尽可能早地在功能描述阶段，就避免使用对资源大量需求的特征曲线和特征图（见本书 5.4.1.5 节讨论）。

下面将讨论软件设计和实现的方法和工具，以及软件函数的数据模型和动态模型。

5.4.1　考虑产品的非功能特性

作为产品的非功能特性的主要例子，首先应该讨论的是，汽车电子系统的成本限制对

软件开发的影响。在根据数值算法设计具体的物理软件函数时，硬件资源的局限性对可用的数值算法选择有很大的影响。基于实例，本节讨论各种优化措施，以降低对硬件资源的需求。数据变量的实现将在数据模型的实现部分讲解。

5.4.1.1 考虑对不同存储部件不同存储时间的实时优化

通常，微控制器的不同存储部件（例如 RAM、ROM 或 Flash 存储器）的存取时间不同，其结果对程序运行时间有影响。设计软件时，可以对需频繁执行的程序段，例如特征曲线的插值算法，存储在存取时间较低的存储单元中，而获得执行时间的优化。应用在各种不同硬件平台的软件组件，可以通过适当的配置选择来满足这一要求。

举例：实时操作系统 ERCOS^{EK} 结构[86]

实时系统 ERCOS^{EK} 通过它们的模块结构，考虑到不同存储部件的不同存储时间。图 5-54 说明存储在在微处理器的不同存储部分，操作系统的单个组件的存储情况。

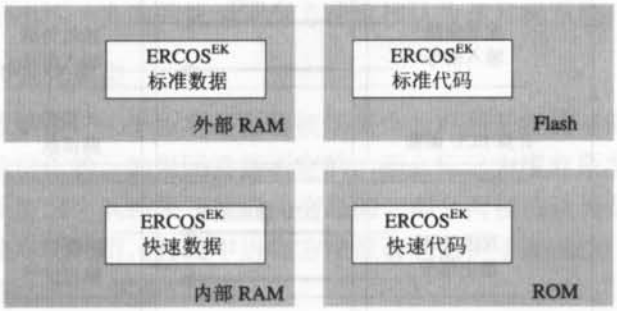


图 5-54 ERCOS^{EK} 实时操作系统的存储器配置[86]

因为来自微处理器的 ROM 的可执行程序，其执行速度比闪存的快，需经常运行的操作系统的代码，通常包含在 ERCOS^{EK} 快速代码部件中且存储在 ROM 中。不常被使用的代码被放在 ERCOS^{EK} 的标准代码组件，且存储于闪存中。根据不同的存储频率，操作系统的数据结构可以被分配在有不同存储时间（如外部或内部）的 RAM 上。

5.4.1.2 通过把软件函数分配到几个任务的实时优化

另一种实时优化措施是把软件函数分成具有不同实时要求的几项任务。子函数执行的必要的采样速率由被控对象的物理特征决定。例如，温度在周围大气慢慢地发生变化，但是内部压力变化是非常快的。在这种情况下，那些依赖周围空气温度变化的子函数将被分配到“慢”任务，而依赖内部压力大小变化的子函数将被分配到“快”任务中。

举例：将软件函数分配到多任务

图 5-55 表明如何把一个软件函数分成子函数 a、b、c，同时函数 a、b、c 被分配任务 A、B、C。这些任务有不同的激活速率，A 的激活速率是 100 ms，B 是 10ms，C 是 20ms。函数 a 计算一个通过 Message X 传给任务 B 的中间变量，并且变量被函数 b 进一步运用。在子函数 c 计算消息 Y，且在任务 B 中也使用消息 Y。只要任务间通信所需要的额外运行时间不超过，在快速任务 B 中的计算而节省的时间，就是整个软件函数减少了的运行时间。

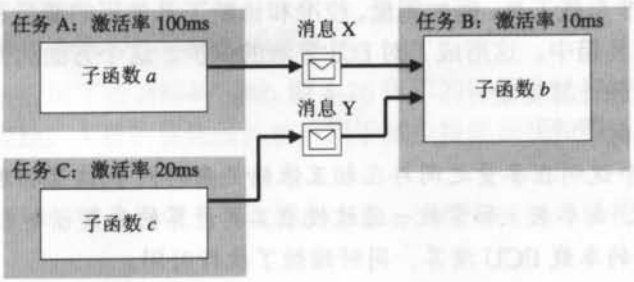


图 5-55　软件功能在不同任务上的分配，以及基于消息的任务间的通信

可以说，只有当整个的资源消耗降低时，优化措施才是成功的。例如，在许多实例中，运行时间减少必然导致必要的存储容量的增大，反之亦然。我们应该牢记这些事实并在实际应用中协调好。

5.4.1.3　通过在线和离线来实现资源优化

对于运行时间优化而言，许多优化也是离线执行的（如先于实际运行）。基于这个原因，在线和离线演算之间的区别是有用的。来自实时操作系统领域一个例子，包括动态在线任务规划和静态离线任务规划，在第 2 章 2.4.4.6 节已经讨论过。

举例：不必要的消息副本的离线优化

另一个例子是由不必要的消息副本的离线优化组成的（见图 2-36）。任务 A、B、C 的优先权在图 5-56 中做了说明。如果一个有优先权的微处理器时序决策被使用，有比较高的优先权 2 的任务 B，可以打断优先级为 1 的低优先级任务 A，对应地，任务 B 也可能被任务 C 打断，因为任务 C 的优先级别为 3。因为任务 C 能改变信息 Y 的值，任务 B 运行的开始时，任务 B 被要求去存储消息 Y 的复制值；因而在整个任务 B 的执行过程中，一直具有相同的消息 Y 的值。由于级别低，任务 A 不可能打断任务 B 的运行，因而在执行的过程中信息 X 不能改变，这样就不必要在任务 B 中复制信息值 X，因此，当了解了预定的策略后，将决定在离线状态下，复制信息是否是必需的。用这种方法，可以避免不必要的信息副本，可以达到节省运行时间和存储容量的效果。

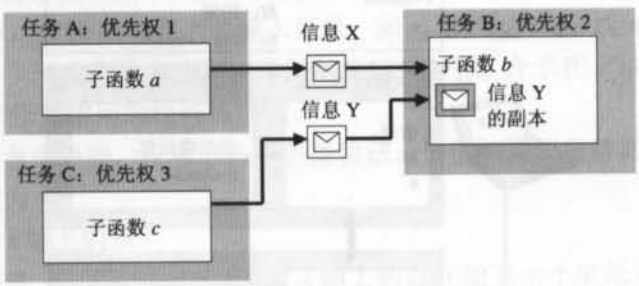


图 5-56　不必要的信息副本的离线优化

5.4.1.4　分成车载和非车载计算的资源优化

另外一种有潜力的优化是通过分成车载和非车载演算来实现的。在车载 ECU 网络上

不要求，仅被用做非车载工具，例如测量、校准和诊断工具使用的变量和计算，可以从 ECU 中被转换到这些工具箱中。这形成了对 ECU 资源的保护。这个方面的例子，是仅要求非车载使用的依赖参数和计算变量。

举例：依赖参数[73, 87]

图 5-57 是一个说明在参量之间存在相互依赖关系的例子，这里物理方程被分成车载和非车载演算。为此，由参数 r 和常数 π 通过校准工具计算的参数 d 和 U 被引入到 ECU 中。这样就降低了必要的车载 ECU 演算，同时缩短了执行时间。

这样做的另一个好处是，当校准工具连续跟踪依赖参量 d 和 U 的值时，参数 r 必须适应校准工具。

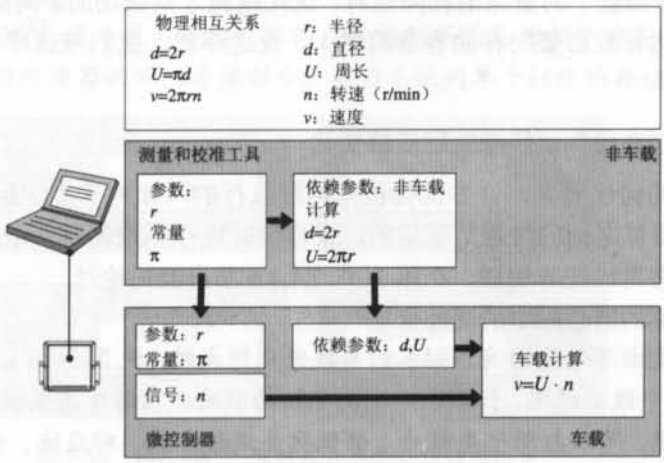


图 5-57 依赖参数

举例：计算参数[87]

类似的优化方法，通过对参数的非车载计算而实现，如图 5-58 所示。转矩信号 M 和转速信号 n 存储在 ECU 中，而 P 由测量工具计算。

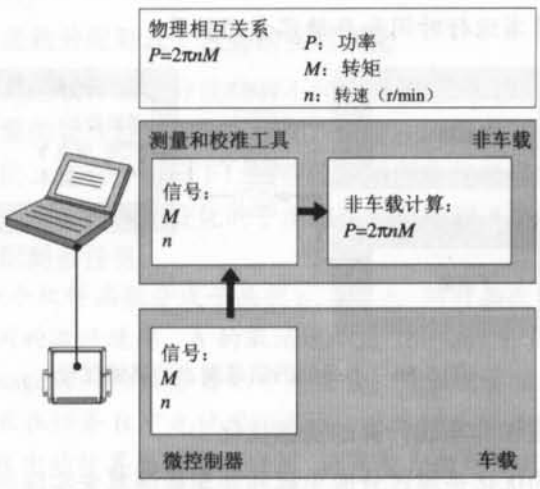


图 5-58 计算参数

5.4.1.5　对于特征曲线和特征图的资源优化

许多优化措施被应用于数据结构领域，图 4-26 所示的特征曲线和特征图形被大量地应用于许多函数。相应地，大量的优化措施被应用于减少特征曲线和特征图的存储容量和运行时间。

下面的例子介绍了特征曲线中几个可行的方法，这些方法也可以相似地被应用于特征图中。

1．数据结构、插值及特征曲线外推法

通常，特征曲线被存放在一个数据结构表中，如图 5-59。表中的第一行是 X 轴方向输入变量的严格单调递增序列。X 轴上的点，记为 x。第二行是各特征曲线对应 X 轴上点的值，记为 y，作为特征曲线的输出量。

可视表格

横轴点 x	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
特征值 y	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8

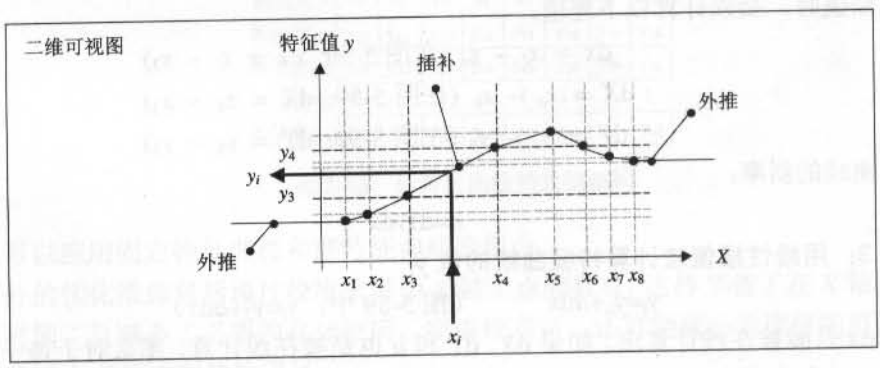


图 5-59　特征曲线的数据结构和插值与外推

特征曲线数据结构的其他元素（如 X 轴上点的数量）是输出值计算的辅助变量。除表格表示外，特征曲线常在一张 x-y 图中可视化表示。

如图 5-59 所示，其他 X 轴点上的输入变量，如 x，$x< x_{\min} = x_1$ 或 $x > x_{\max} = x_8$，通常被外推。图 5-59 描述了一个在许多应用中常用的恒定值外推。介于在 X 轴两点之间的输入值被内插，图 5-59 所示为常用的线性插值法。

下面的讨论不考虑外推，仅考虑内部的插值法。插值法算法通常通过三步来确定输入值为 x_i 时，输出值为 y_i。

步骤 1：查找 x 轴上的点

相对于输入变量 x_i，确定一个相邻的 X 轴上的点（如下一个更低的或更高的点）。如图 5-60 中，下一个更小的到 x_i 的点 x_u 即为 x_3。由于 X 轴上的点是严格单调的，所以下一个更高的 X 轴点为 $x_o = x_4$。

对于 X 轴上的点，可以用几种搜索算法。例如：从 X 轴上最后的有效值开始搜索、二分法搜索，或从 X 轴上的最大值开始搜索，反之亦然（图 5-60）。

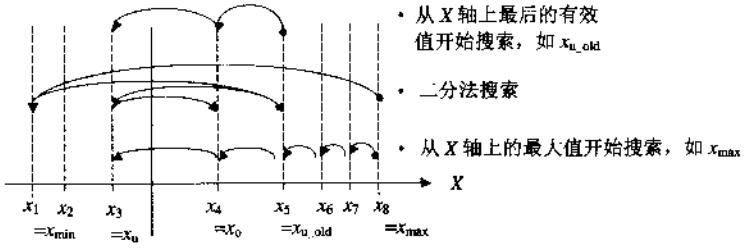


图 5-60 X 轴不同点的寻找方法

恰当的搜索方法取决于 X 轴上点的分布和应用。如发动机 ECU 的运行时间比出现高的发动机速度的时间要短得多。因此，以转速输入的特征曲线情况下，它会有意识地由高转速搜索到低转速。因而，以更低的转速换取更长的搜索时间也没关系。

运行时间有限的某些应用案例中，特征曲线的 $y_u=y(x_u)$或 $y_o=y(x_o)$可以作为输出变量的值。更加准确的值可以通过插值算法获得。实际中，在 X 轴两点之间通常采用线性插值法。

步骤 2：计算斜率 *a*

线性插值时，必须计算以下差值：

$$dx = x_i - x_u \text{ (在图 5-59: } dx = x_i - x_3) \tag{5.8}$$

$$dX = x_o - x_u \text{ (在图 5-59: } dX = x_4 - x_3) \tag{5.9}$$

$$dY = y_o - y_u \text{ (在图 5-59: } dY = y_4 - y_3) \tag{5.10}$$

特征曲线的斜率：

$$a=dY/dx \tag{5.11}$$

步骤 3：用线性插值法计算特征曲线的值 y_i

$$y_i=y_u+adx \quad \text{（图 5-59 中：} y_i=y_3+adx\text{）} \tag{5.12}$$

变量 dx 只能被在线计算出。如果 dX、dY 和 a 也是被在线计算，那么对于每一个插值，三个减法、一个除法、一个加法和一个乘法都必须在线进行。

或者可以通过离线和非车载时计算 dX、dY 的值。这个选择是以优化为目的的。以下的方法被广泛应用。

2. 以一种扩展的特征曲线数据结构存储斜率 *a*

使用插值法时，以一种扩展的特征曲线数据结构存储斜率 a 可以节省在线计算时间。结果，只有一个减法、一个加法和一个乘法必须在线被计算。在对时间敏感的应用中，必需的存储量的增加是完全可以接受的（图 5-61）。

横轴点 x	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
特征值 y	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
斜率 a	a_1	a_2	a_3	a_4	a_5	a_6	a_7	

图 5-61 特征曲线的带有存储斜率 a 的扩展数据结构

3. 固定的特征曲线

X 轴上两点的距离 $dX = x_o-x_u$ 被定义为常数，这样 x_u 不必通过搜索就可以被算出。离

线和非车载情况下，dX 的倒数可以被求出，并与特征曲线数据存放在一起。这时，插值法不再要求除法，但是在线计算中必须有二个减法、一个加法和二个乘法。

X 轴上分布的点可以根据极小点 x_{min} 的值被定义和存放，X 轴上点的数量为 n，X 轴点间的距离为 dX。这种数据结构特别适合为特征曲线存放大量的 X 轴点（图 5-62）。

最小的横轴点 x_{min}	x_{min}							
轴点间距 dX	dX							
1/dX	$1/dX$							
轴点计数 n	n							
特征值 y	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8

图 5-62　固定特征曲线的数据结构

4. 群特征曲线

因为各种特征曲线有相同输入变量，一致的 X 轴点分布被指定，X 轴上点的搜索和差别的计算只需一次，每一次都要通过输出变量 y_i 的计算完成插值法（图 5-63）。

横轴点 x	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
特征值 y_1	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}	y_{17}	y_{18}
特征值 y_2	y_{21}	y_{22}	y_{23}	y_{24}	y_{25}	y_{26}	y_{27}	y_{28}
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
特征值 y_n	y_{n1}	y_{n2}	y_{n3}	y_{n4}	y_{n5}	y_{n6}	y_{n7}	y_{n8}

图 5-63　群特征曲线的数据结构

也可以应用固定特征曲线和群特征曲线的组合。

另外的优化措施包括通过校准来减少 X 轴上点的数目，这样节省了在 X 轴点搜索期间的计算时间，且减小了必要的存储空间。完成校准后，还可能减小物理值的范围，增加了量化，或减少 X 轴点和值的码长。

用于特征曲线 X 轴上的点和对应的值在处理器中内部表示的，码长的许多可能的组合方法，产生了大量不同的对存储量要求高的插值方法。通过限制允许特征曲线和特征图的存储结构，可以减少可能的组合的数量。

5.4.2　定点和浮点算法的设计及实现

这部分着重讲解定点和浮点算术的算法设计。重点讨论当今实际在线应用（即在微控制器 ECU 中使用）的基本方法。除基本的算术操作之外，在线应用主要包括特征曲线和图形的插值法，信号的数值微分和积分方法，以及数字滤波过程。例如，包含这些函数的基本模块库是在 MSR-MEGMA 项目中被标准化的[79]。然而，本节仅仅讨论几个在设计和应用中的基于机器算法的基本问题。

5.4.2.1　数字处理器中数的表示

所有的数字处理器中的数都是二进制形式的，其中二进制项系数 a_i 被用来表示 x：

$$x = \pm(a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + \cdots) \tag{5.13}$$

举例：x=9 的二进制表示

将 x=9 分解，得 $9=1\times2^3+0\times2^2+0\times2^1+1\times2^0$

所以其二进制表示为 1001。

考虑到十进制和二进制的区别，二进制表示在下文中以黑体字表示。

对于数的内部表示法，数字处理器仅提供的二进制位为固定长度为 n，即码长，由处理器结构决定，且可以扩展成 n 的整数倍（如 $2n$，$3n$）。相应地，8 位码长的微处理器称为 8 位微处理器，16 位码长的称为 16 位微处理器，以此类推。

用 n 位码长表示数字有几种方式。

1. 定点表示法

对于 $n=n_1+n_2$，不但数字 n 是固定的，而且小数点前后的数字 n_1 和 n_2 也是固定的。通常，$n_1=n$ 或 $n_1=0$。

目前，ECU 使用的许多微处理器的功能仅限于对数字的定点表示和处理。

没有一般有效性约束，以下讨论中，假定定点表示法中 n_1=n。因而，数字 n 决定数组的表示。

例如，当 n=8 时，数字 0 至 255 能由 0000 0000 到 1111 1111 的二进制码表示。相应地，这是由 8 位表示的无符号整型数，简记为 uint8。

可以利用一个标志编码位（即标志位）来表示负数。因而，当 n=8 时，数字–128 至 0，0 至+127 都可以表示出来。相应地，这是由 8 位表示的有符号整数，简记为 sint8。

同样，数字的 16 位和 32 位表示方法分别记做 uint16、sint16、uint32 和 sint32，取值范围见表 5-2。

表 5-1 整数的简写符号及取值范围

二进制位数	简 写	取值范围
8 位无符号整数	uint8	0…255
8 位有符号整数	sint8	–128…127
16 位无符号整数	unit16	0…65 535
16 位有符号整数	sint16	– 32 768 …32 767
32 位无符号整数	unit32	0… 4 294 967 295
32 位有符号整数	sint32	– 2 147 483 648 … 2 147 483 647

2. 浮点表示法

小数点的位值与给定数的值一致。因此，每一个数都提供表明小数点位置的信息（即第一个数字后的位数），可以利用指数实现小数点的浮动，这样，实数 x 可以被表示成积的形式：

$$x=a\times2^b \tag{5.14}$$

这里$|a|<1$，且 b 为整数。指数 b 表明尾数 a 中小数点的位置。

举例：x=9.5 的二进制表示

因为 $9.5=1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}$

所以 x 的二进制表示为 1001.1 或 0.10011×2^{100}

与定点表示法相似，对于浮点数表示法，数字处理器仅仅提供定长有限的二进制位，即表示尾数 a 和指数 b 的数 m 和 e，这里 $n=m+e$。

ECU 是否装备能够支持浮点表示法和数字处理功能的微处理器，取决于各自的应用需求。

数字的浮点表示法不唯一。上面的例子中，也可以表示为 0.010011×2^{101}。因此，尾数 a 第一数字不为 0（非零)的任一个数字的浮点表示法称作规范式。在二进制下，$|a|\geqslant2^{-1}$。因而，除了第一个零，尾数 a 的所有数字都是有效数字。

不失一般性，以下的讨论假定在标准浮点表示法且可以进行浮点计算的基础上进行。

数字 m 和 e，以及表示法中基数 $B=2$，则在机器中可以确定数集 A。集合 A 是实数集 $\mathbf{R}$ 的子集($A\subseteq\mathbf{R}$)。A 中的元素称做机器数。

在 IEEE 标准中，定义了 $n=32$ 和 $n=64$ 时的浮点表示法。与定点数相似，32 位和 64 位浮点数分别被命名为 real32 和 real64。

由于用定点和浮点表示的数集 A 是有限的，软件综合行为的设计和应用，提出这样一个直接的问题，如何用一个机器数逼近一个非机器数（$x\notin A$）。这个问题不仅体现在计算机中输入数据时，也体现于处理器对内部数据处理过程中。

可以用简单的例子说明，尽管 a、b 为机器数（即 $a,b\in A$）但是通过简单的运算（如加法 $a+b$、减法 $a-b$、乘法 $a\times b$ 和除法 a/b）得到的结果 c 却不属于 A。

因此，机器数 rd(x)（rd(x)$\in A$）对非机器数 x ($x\notin A$)的逼近，有如下要求：

$$|x-\mathrm{rd}(x)|\leqslant|x-g_k| \tag{5.15}$$

$\forall\, g_k\neq\mathrm{rd}(x)$且 $g_k\in A$（如图 5-64 所示）。

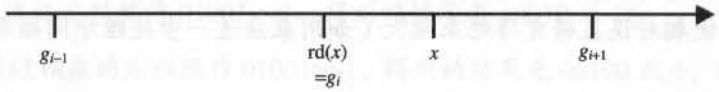


图 5-64　用 rd(x)逼近 x

通常可以通过取整或者在上溢或下溢处理的极限内来确定 rd(x)，下面将通过几个简单的例子讨论取整误差，以及上溢和下溢处理。这里将以获得可能的最准确的结果作为优先目标。

在定点表示法中，所有数会发生取整误差、上溢和下溢。为了能更清楚地讲解，下面的例子中主要基于数字的 uint8 表示。

5.4.2.2　在整数除法时的取整误差

因为 a/b 的准确运算结果可能不是整数，整数除法 $c=a/b$ 可能产生取整问题。

举例：整数除法和取整法

变量 a、b 和 c 均以 uint8 记法表示。

除法　$c=a/b$，其中 $a,b,c\in A=\{0,1,2,\cdots,255\}$，有以下几种情形：

情形 1：$a=100$，$b=50\rightarrow c=2\in A$

情形 2：$a=19$，$b=2\rightarrow c=9.5\notin A$

情形 3：a=240，b = 16→ c =1.49∉A

情形 4：a=100，b = 201→ c =0.49∉A

情形 5：a=100，b = 1→ c =100∈ A (价值不大)

情形 6：a=100，b = 0　被 0 除，无意义

在情形 1 中，得到的结果是整数，可用 uint8 数字表示，不会发生取整错误。

在情形 2 中，得到的结果不是整数，因此必须进行取整。c 的二进制表示为：

$$c = (a_n\times 2^n + a_{n-1}\times 2^{n-1} + \ldots + a_0\times 2^0 + a_{-1}\times 2^{-1} + a_{-2}\times 2^{-2} + \cdots)$$

所以，c =9.5=1 × 23 + 0 × 22 + 0 × 21 + 1× 20 + 1× 2−1 或 c=1001.1，因此，a_3=1，a_2=0，a_1=0，a_0=1，a_{-1}=1。

取整值 rd(c)的形式为：

$$\text{当 } a_{-1}=0 \text{ 时，} \mathrm{rd}(c)= a_n\, a_{n-1}\cdots a_0 \tag{5.16}$$

$$\text{当 } a_{-1}=1 \text{ 时，} \mathrm{rd}(c)= a_n\, a_{n-1}\cdots a_0 +1 \tag{5.17}$$

所以，在这个情形中，取整值 rd(c)=10 或 1010。

许多微处理器中不用取整法，而使用整数除法（即小数点后的部分被简单地删除），结果往往是整数。

对于任意的 a_{-1}：

$$\mathrm{rd}(c)= a_n\ a_{n-1}\cdots a_0 \tag{5.18}$$

所以，在这种情形下，rd(c)=9 或 1001。

在情形 3 中，用取整法和整数除法，均可得到 rd(c)=1。然而，在这种情形下，相对取整误差 ε =[rd(c)−c]/c = (1−1.49)/1.49≈−1/3(这个误差说明结果的准确性)已经变得相当大。

在情形 4 中，得到 rd(c)= 0。这里，相对取整误差(0−0.49)/0.49=−1 特别大。这种情形下误差的积累将使相对误差将变得越来越大（如用乘法进一步处理中间结果时，会积累误差）。

在情形 5 中，除数为 1，可以忽略。

在情形 6 中，除数为 0，在算法中无意义且必须被作为异常进行处理。

对于 c>1，相对取整误差ε的计算如下：

$$|\varepsilon| = \left|\frac{\mathrm{rd}(c)-c}{c}\right| \leqslant \frac{1}{3} \tag{5.19}$$

对于 c>1，下列适用于相对截断误差：

$$|\varepsilon| = \left|\frac{\mathrm{rd}(c)-c}{c}\right| \leqslant \frac{1}{2} \tag{5.20}$$

所以，当 c>1 时，rd(c)= c(1+ε)，且相对取整误差|ε|≤1/3，相对截断误差|ε|≤1/2。因而，相对取整误差比相对截断误差稍小。在两种情形下，相对误差会随着结果的增大而减小。

5.4.2.3　加法、减法和乘法中的上溢和下溢

在定点整数表示法中，如果操作数 a，b∈A 可以当做机器数使用，那么通过基本运算，如加法 a+b、减法 a−b 和乘法 a×b，得到的结果是整数，不会发生取整误差。但是，由于二进制的位数 n 有限，所以总是会有一些非机器数的 x∉A。

举例：加法、减法和乘法

变量 a、b、c 以 uint8 记法表示。

加法 $c=a+b$，且 $a,b,c \in A=\{0,1,2,\cdots,255\}$，有如下情形。

情形 1: $a=100,b=100 \rightarrow c=200 \in A$

情形 2: $a=100,b=157 \rightarrow c=257 \notin A$

减法 $c=a-b$，且 $a,b,c \in A=\{0,1,2,\cdots,255\}$，有如下情形。

情形 3: $a=100,b=100 \rightarrow c=0 \in A$

情形 4: $a=100,b=102 \rightarrow c=-2 \notin A$

在情形 2 中，结果 c 太大而不能用 uint8 整数表示，称做上溢。

在情形 4 中，结果 c 太小而不能用 uint8 整数表示，称做下溢。

在乘法中也可能发生类似的情形。

浮点算法中也会产生类似的误差。但是，在浮点算法中产生的取整误差要比定点算法中的小几个数量级。

5.4.2.4　平移操作

由于在处理器中使用二进制，只要操作数 $b \in \{2^1,2^2,\cdots,2^n\}$，那么通过平移操作可以高效地进行乘法 $a \times b$ 和除法 a/b 运算。

举例：平移操作

$x=9$ 可以分解成如下的形式：

$9=0\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0$

所以其二进制表示为 01001。

9×2 可以通过左移操作 01001<<1，得到的结果是 10010 或 18。

$\frac{9}{2}$ 可以通过相应的右移操作 01001>>1，得到的结果是 00100 或 4，因而，右移操作截断了小数位。

注意到，对于有符号整数，如 sint8、sint16、sint32，某些情形下通过右移操作符>>，标志位本身成为数字表示法中的一个正常数字。因此，应认真检查，在这种情形下正规除法是否更好。

5.4.2.5　处理上溢和下溢

对于因上溢或下溢导致数值超出取值范围的处理，依赖于处理器。在算法中可以有不同的处理方法。以加法为例，下面将对经常应用的上溢处理方法进行讨论。

举例：上溢处理

变量 a，b 和 c 用 uint8 表示。

加法 $c=a+b$，且 a，b，$c \in A=\{0, 1, 2, ..., 255\}$，得到如下结果：

情形 1：$a=100$，$b=100 \rightarrow c=200 \in A$

情形 2：$a=100$，$b=157 \rightarrow c=257 \notin A$

对情形 2 中的上溢，可能有以下的处理方法：

（1）有检测的或无检测的上溢：允许上溢。多数微处理器的输出 $c=a+b-256=1$。

算法中，通常使用比较操作$(c<a)$ && $(c<b)$检测和处理无符号整型上溢。

（2）限制结果：在算法中识别出上溢，且结果 c 不超过最大的可表示值 255。

（3）扩大结果的数值范围：结果 c 可以在一个扩大的数值范围（如 uint16 或 sint16 中）中被表示出来，因而，当 a，$b\in A_{\text{uint8}}=\{0, 1, 2, ..., 255\}$时，$c=a+b$，得到

$$c\in A_{\text{uint16}}=\{0, 1, 2, \cdots, 65535\}$$

或

$$c\in A_{\text{sint16}}=\{-32768, \cdots, 0, 1, 2, \cdots\cdots, 32767\}$$

c 是一个 sint16 型的变量，不再发生上溢，且减法中也不再发生下溢。

（4）调整结果。检测出上溢，且将结果 c 调整为 rd(c)。为此，引入 c 的分量 q，且$|q|>1$。通过调整 c 为方程 $c=q\times\text{rd}(c)$，能够扩大 c 的数值范围，且不再发生上溢。可以通过平移操作，实现在集合$\{2^1, 2^2..., 2^n\}$中调整因子 q。因而，$\text{rd}(c)=(a+b)/q$，这里 $a, b, \text{rd}(c)\in A_{\text{uint8}}=\{0, 1, 2, ..., 255\}$，且 $q=2$，$c\in A_c=\{0, 2, 4, \cdots, 510\}$，不再发生溢出。其相对误差 ε 为：

$$|\varepsilon|=\left|\frac{\text{rd}(c)-c}{c}\right|\leqslant\frac{q-1}{c} \tag{5.21}$$

与上面的例子相似，相对误差随运算结果的增加而减小。

5.4.2.6 定点算法中运算的误差传播

现在主要研究给定算法中的误差传播方式，首先，我们给出“算法”的准确定义。下面的讨论将算法定义为：具有明确定义的按一定次序的有限的简单操作序列，特定的输入数据通过这种操作的计算，能得到问题的解。

举例：算法的定义

定义表达式：$d=a+b+c$

尽管 $d=(a+b)+c$ 和 $d=a+(b+c)$在数学上等价，但它们在定点计算中可能产生不同结果。

算法一步骤如下：

- Step 1.1： $\eta_1=a+b$
- Step 1.2： $d=\eta_1+c$

算法二步骤如下：

- Step 2.1： $\eta_2=b+c$
- Step 2.2： $d=a+\eta_2$

这里 a,b,c,d 以 sint8 整数形式表示。这样，

$$a, b, c, d\in A=\{-128, \cdots, +127\}$$

上溢和下溢能够被检测出来，且将其值限制到–128 ... +127 间。

将 $a=101$，$b=-51$，$c=-100$ 代入，得：

算法一：$\eta_1=a+b=101-51=50$

$d=\eta_1+c=50-100=-50$

算法二：$\eta_2=b+c=-51-100=-128$（下溢限制）

$d=a+\eta_2=101-128=-27$

不同的算法通常产生不同的结果，究其原因，主要是由取整和限制误差的传播引起的。因此，为提高运算的准确性，必须制定相应的准则。

在定点运算中，d 的值由其近似值 rd(d)代替。对于算法一，$\mathrm{rd}_1(d)$为：

$$\mathrm{rd}(\eta_1)=(a+b)(1+\varepsilon_{1.1}) \tag{5.22}$$

$$\mathrm{rd}(d_1)=(\mathrm{rd}(\eta_1)+c)(1+\varepsilon_{1.2})=\left[(a+b)(1+\varepsilon_{1.1})+c\right](1+\varepsilon_{1.2})$$

$$=(a+b+c)\left[1+\frac{a+b}{a+b+c}(1+\varepsilon_{1.2})\varepsilon_{1.1}+\varepsilon_{1.2}\right] \tag{5.23}$$

因此，相对误差 ε_{d_1}：

$$\varepsilon_{d_1}=\frac{\mathrm{rd}(d_1)-d}{d}=\frac{\mathrm{rd}(d_1)}{d}-1=\frac{a+b}{a+b+c}(1+\varepsilon_{1.2})\varepsilon_{1.1}+\varepsilon_{1.2} \tag{5.24}$$

正如第一个近似，当忽略高阶项如$\varepsilon_{1.1}$、$\varepsilon_{1.2}$，算法一产生的误差为：

$$\varepsilon_{d_1}\approx\frac{a+b}{a+b+c}\varepsilon_{1.1}+1\cdot\varepsilon_{1.2} \tag{5.25}$$

同样，算法二产生的误差为：

$$\varepsilon_{d_2}\approx\frac{b+c}{a+b+c}\varepsilon_{2.1}+1\cdot\varepsilon_{2.2} \tag{5.26}$$

放大因子为 $\frac{a+b}{a+b+c}$ 与 1，和 $\frac{b+c}{a+b+c}$ 与 1，这表明中间结果的取整误差对相对误差 ε_d 有很大影响，其中关键因素是 $\frac{a+b}{a+b+c}$，和 $\frac{b+c}{a+b+c}$。这取决于(a+b)与(b+c)哪个更小，来决定用公式(a+b)+c，或 a+(b+c)来计算的 a+b+c，在“数值上更稳定”。

在上面的例子中，a+b=50，b+c=−151，因为取值范围的限制，$\varepsilon_{2.1}$ 特别大：

$$\varepsilon_{2.1}=\frac{(-128+151)}{(-151)}\approx-0.15$$

而$\varepsilon_{1.1}$，$\varepsilon_{1.2}$和$\varepsilon_{2.2}$都为 0。

因此，误差为：ε_{d2}=[(−151)/(−50)]×(−0.15)= −0.45。

因而，这说明，尽管第二步计算没有相对误差，但是，将第一步运算中的相对误差$\varepsilon_{2.1}$，带入到算法二中得到的放大因子约为 3。这就可以解释，当考虑到例子中的输入值时，算法一在数值上更优。

尽管这种方法可以建立系统化的算法，但通常比较麻烦。对于很少的取整误差的影响也必须加以估计。然而在典型的算法中，以该方式确定所有的取整误差的影响时，算法的操作数目和取整误差的数目太大。这些情形用其他方法可能会更好，其中之一就是中间计算[75]，本节我们将对此不加以详细地讨论。下面的例子，仅限于讨论在给定算法的第 i 步中的相对误差ε_i。

举例：算法三

上例中算法二中的中间结果通过因子 2 加以调整，产生算法三：

- Step3.1：$b_1=\frac{b}{2}=-\frac{51}{2}=-25,\varepsilon_{3.1}=\frac{[-50-(-51)]}{(-51)}=-\frac{1}{51}$
- Step3.2：$c_1=\frac{c}{2}=-\frac{100}{2}=-50,\varepsilon_{3.2}=0$
- Step3.3：$\eta_2=b_1+c_1=-75,\varepsilon_{3.3}=0$

- Step3.4：$a_1=\frac{a}{2}=\frac{101}{2}=50,\varepsilon_{3.4}=\frac{[100-101]}{(101)}=-\frac{1}{101}$
- Step3.5：$d_1=a_1+\eta_2=50-75=-25,\varepsilon_{3.5}=0$
- Step3.6：$d=d_1\times2=-25\times2=-50,\varepsilon_{3.6}=0$

带入这些调整数据后，产生的结果比在算法二中具有中间结果限制得到的更为准确。这类中间结果限制可能隐含于算法中，所以必须引起注意（例如子程序调用中）。

5.4.2.7 物理关联和定点运算

经常出现这样的情形，微处理器产生的取值范围不同的两个物理信号，必须在一个数学运算中处理。下例讨论两个信号相加，多个操作数的运算可分解成多个两个操作数的运算。

举例：具有不同量化关系的的两个信号的加法

一个简单的例子是两个信号 a 和 b 的加法，目的是实现物理关系 $c_{phys}=a_{phys}+b_{phys}$。

微处理器中所用的信号 a、b、c，是用 unit8 表示的定点变量 a_{impl}，b_{impl}，c_{impl}。

连续物理变量和以离散点表示的实现变量是一种线性关系，有上限和下限，图 5-65 表明了变量 a 之间的关系。

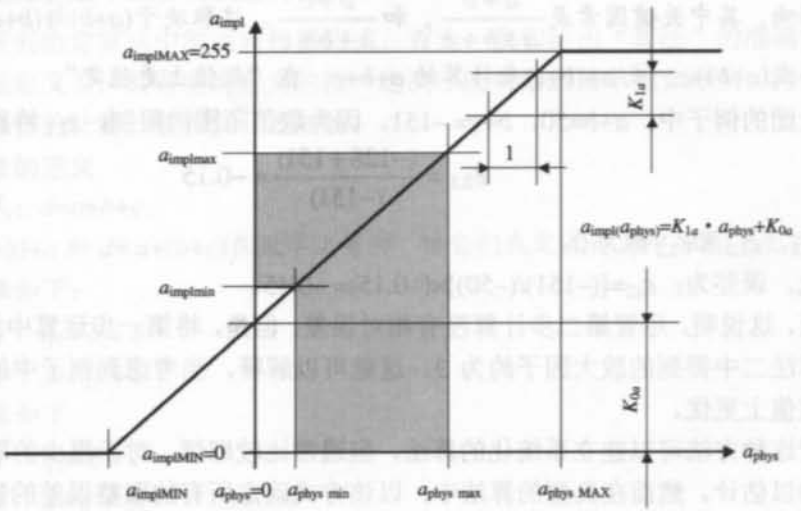


图 5-65 物理变量与实际变量的关系

关于这些限制，我们必须注意实现层的数值表示引起对应物理实际值的变化。在图 5-65 中，a_{impl} 以 unit8 表示，其取值范围为{0,1,2,…,255}，或通常的值的范围{$a_{impl\ MIN}$,…,$a_{impl\ MAX}$}。相应地，此例中得到的物理表示的值的上、下限分别为：

$$a_{physMIN}=\frac{(a_{implMIN}-K_{0a})}{K_{1a}}=\frac{(-K_{0a})}{K_{1a}} \tag{5.27}$$

$$a_{physMAX}=\frac{(a_{implMAX}-K_{0a})}{K_{1a}}=\frac{(255-K_{0a})}{K_{1a}} \tag{5.28}$$

实现层上值的取值范围{$a_{phys\ min}$ … $a_{impl\ max}$}不能与物理产生的值的上限 $a_{phys\ MAX}$ 和下

限 $a_{\text{phys MIN}}$ 混淆。

同样的关系可以应用于变量 b 和 c，对于线性范围，应用如下：

$$a_{\text{impl}}(a_{\text{phys}}) = K_{1a} \cdot a_{\text{phys}} + K_{0a} \tag{5.29}$$

$$b_{\text{impl}}(b_{\text{phys}}) = K_{1b} \cdot b_{\text{phys}} + K_{0b} \tag{5.30}$$

$$c_{\text{impl}}(c_{\text{phys}}) = K_{1c} \cdot c_{\text{phys}} + K_{0c} \tag{5.31}$$

因为在应用层只能用定点数表示，因此，图 5-65 中，每种情形都要求取整，但图中被忽略了。$\frac{1}{K_{1i}}$ 也称为量化或分解，且 K_{0i} 称为补偿。

实现层中物理变量的加法在下面的算法中实现：

第一步：从 a_{impl} 和 b_{impl} 中减去补偿。

$$a_{\text{impl_1}} = a_{\text{impl}} - K_{0a} \tag{5.32}$$

$$b_{\text{impl_1}} = b_{\text{impl}} - K_{0b} \tag{5.33}$$

第二步：对 $a_{\text{impl_1}}$ 改变量化，得到 $a_{\text{impl_2}}$。

$$a_{\text{impl_2}} = a_{\text{impl_1}} \cdot \frac{K_{1b}}{K_{1a}} \tag{5.34}$$

第三步：加法。

$$c_{\text{impl_1}} = a_{\text{impl_2}} + b_{\text{impl_1}} \tag{5.35}$$

第四步：对 c_{impl} 改变量化，得到 $c_{\text{impl_2}}$。

$$c_{\text{impl_2}} = c_{\text{impl_1}} \cdot \frac{K_{1c}}{K_{1b}} \tag{5.36}$$

第五步：对 $c_{\text{impl_2}}$ 进行补偿，得到 c_{impl}

$$c_{\text{impl}} = c_{\text{impl_2}} + K_{0c} \tag{5.37}$$

另外一种方法是，第二步后的计算基于 a_{impl} 的量化值，在这种情形下，依据 a_{impl} 的量化值来改变 b_{impl} 的量化值，第四步可以做相应的改变。

第三种选择是从第二步起的计算直接依据 c_{impl} 的量化值。

正确地选取这三种方法中之一，可以使需要再次量化的数目最小。

如果能通过熟练地选择 K_{1a}、K_{1b} 和 K_{1c} 进行量化，那么可以通过移位操作实现必要的转换。这样，建议以 $\frac{K_{1b}}{K_{1a}}$，$\frac{K_{1c}}{K_{1b}}$ 的方式来选择量化，且取值于集合 $\{2^1, 2^2, \cdots, 2^n\}$。如果选择了同样的量化方法，步骤二与/或步骤四可以省略。

通过对应 $a_{\text{impl MIN}}$ 和 $a_{\text{impl MAX}}$ 的中间值计算，可以用于预先确认中间结果是否在所选的操作数的取值范围 $a_{\text{impl MIN}}$ 和 $a_{\text{impl MAX}}$ 中。通过更正参数值 K_{1i} 和 K_{0i}，使中间数值更合适，进而提高计算结果的准确性。

只要在更大的取值范围进行中间计算，就可以避免限制和溢出处理。

将在线和离线计算分开可得到进一步优化，如方程（5.34）和（5.36）中的除法可通过离线计算。为了清楚地说明，这类优化在上个例子中被有意忽略。

5.4.2.8 物理层模型和实现层

如前例所示，区分物理层和应用层在算法上很有意义。原因是，通过区分不同的层，

物理关联与微处理器上实现的详细情形，如量化方法的选择，字长和整数运算的方法等，被分开讨论。在模型的物理层，可以区分为连续值、离散值和布尔变量。

（1）在大多数情形下，连续变量表示物理信号的连续值，如温度、转速或压力。

（2）离散变量表示自然变量，如发动机中的汽缸数目或转换过程中的进程（转换级别）数。

（3）布尔变量描述状态对，如当一个转换位置到相应的另一个赋值状态（开/关，高/低，真/假）。

如果连续变量用定点数表示，首先要将其离散化。因为，在数据建模中，数值离散化通常有重大意义。

这就意味着，每个 X_{phys} 必须被准确地分配一个唯一的离散值 X_{impl}。

X_{impl} 属于集合$\{X_1, X_2, X_3, \cdots, X_n\}$，且 $X_{impl\ min} \leqslant X_{impl} \leqslant X_{impl\ max}$ （5.38）

这种转换通常通过一个转换公式来描述，且在物理模型层或应用层对最小值和最大值进行说明。

在软件组件的开发过程，要求从物理模型层转换到应用层，而在随后的开发过程中，测量 ECU 的内部变量（包括生产和服务阶段），需要从实现变量转到物理单元。

5.4.2.9 定点算法实现时的一些技巧

算法结果优劣的决定因素是相对误差。如前一节所示，整数除法和溢出、下溢处理限制了数的准确性。从这些事实中，可以得到实现中的一些规则和指导准则。

1. 关于整数除法有用的信息（方法）

（1）因为在整数除法中相对误差比较大，所以要尽可能地避免使用它。

（2）零作除数没有意义，因此必须作为异常处理，也可以通过限制或查询将其排除。

（3）对于无符号整数，除数取自集合$\{2^1,2^2,\cdots,2^n\}$，通过移位操作可以高效执行除法运算。

（4）如果算法中必须使用除法运算，那么应该尽可能晚地使用，这样，相对误差才会较晚被带入结果。

（5）整数除法运算的结果越大，相对误差越小。因此，分子应该比分母相对大一些。如果可能，可以通过定义一个补偿，或通过在除法操作之前使用平移操作重新量化，来实现分子比分母大得多的目的。毫无疑问，原来的补偿或量化必须在算法中再次建立。

举例：计算除法 $c_{phys}= a_{phys}/b_{phys}$

变量 a_{impl} 和 *temp* 以 uint16 表示，变量 b_{impl} 和 c_{impl} 以 uint8 表示。

物理值如下：

$a_{phys}=79$，$b_{phys}=5$

c_{phys} 的准确值应该是 $\frac{79}{5}=15.8$。

转化成如下方程：

$$a_{impl}(a_{phys})=K_{1a}\cdot\ a_{phys}+K_{0a}=1\cdot\ a_{phys}+0 \tag{5.39}$$

$$b_{impl}(b_{phys})=K_{1b}\cdot\ b_{phys}+K_{0b}=1\cdot\ b_{phys}+0 \tag{5.40}$$

$$c_{impl}(c_{phys})=K_{1c}\cdot\ c_{phys}+K_{0c}=1\cdot\ c_{phys}+0 \tag{5.41}$$

取值范围如下：

$$a_{\text{impl min}}=0,\ a_{\text{impl max}}=255$$

$$b_{\text{impl min}}=2,\ b_{\text{impl max}}=10$$

为了计算 $c_{\text{phys}}=\frac{a_{\text{phys}}}{b_{\text{phys}}}$，选择下面的算法：

第一步：将 a_{impl} 左移 8 位，利用整 16 位的取值范围 $a_{\text{impl}}=a_{\text{impl}}<<8=a_{\text{impl}}\cdot 2^8$。

因为，$a_{\text{phys}}=79$，所以 $a_{\text{impl}}=79\cdot 2^8=20\,224$。

第二步：进行实际的整数除法：

$$temp=\frac{a_{\text{impl}}}{b_{\text{impl}}}$$

因为 $b_{\text{phys}}=5$，所以得：

$$temp=\frac{20\,224}{5}=4\,044$$

即 $15.7968\cdots\times 2^8$，与 $\frac{79}{5}=15$ 相比，扩展变量 *temp* 的范围有助于提高准确率。

第三步：通过 8 小数位重新调节运算结果。

执行算法时，根据 c_{impl} 的范围重新调整变量 *temp*：

$$c_{\text{impl}}=temp>>8$$

这样可能产生相对误差且使准确率降低，因此在算法中应尽可能晚地执行此步骤。此步之前，每一个计算步骤都应该使用相对准确的中间变量 *temp*。

2. 加法、减法和乘法的要点

（1）溢出和下溢处理限制了加法、减法和乘法的准确性；

（2）有几种可用的溢出和下溢处理方法，如重新调节范围、限制或通过类型转换拓展值的范围等。

（3）即使不发生上溢和下溢，在整个数值域里，重新调节值的范围能减小相对误差。

（4）只有发生上溢或下溢，限制取值范围能降低相对准确率。

（5）利用实际变量中物理信号的转换关系，设置补偿值：在所选的取值范围的中间位置进行实现层的计算，这也有助于在处理器的内部描述中使用较短的字长。在大型的数据结构中，这一优势特别明显，如对于带有补偿的特征曲线、特征图，且以较小的内存需求的形式展现出来。然而，当连接不同的信号时，补偿可能导致额外的转换操作。除了特征曲线、特征图和其他几个情形外，建议一个较好的做法就是转换公式中尽可能避免补偿。

（6）通过移位操作，从集合$\{2^1, 2^2, \cdots, 2^n\}$中取值能有效地完成乘除法运算。应该尽可能地避免对有符号整数进行右移操作。在这种情形下，建议使用常规的除法运算。

3. 关于误差传播的要点

（1）即使运用准确的运算，如加法、减法和乘法运算，输入变量的相对误差也可能很快地被放大。

（2）在误差传播中，应特别注意变量取值的范围，因为变量的取值可能会由于子程序函数的调用而隐式地受到影响。同时应该评价变量的取值对于中间结果的影响。

5.4.2.10 浮点算法实现时的一些技巧

用浮点算法实现时也必须注意：对于浮点数机器中数集 A 也是有限的，在运算操作中不可避免地会出现取整误差，如定点算法一样，在这里不满足结合律和分配律，因为精确的运算操作是由浮点数的运算来近似的。

甚至，给定的事实是：通过浮点数的运算并不能解决所有的数值问题，但是数值范围的扩大确实能够体现出如下优势：降低数值取整误差的影响以及上溢、下溢，从而，在大多数情形下使误差可以被忽略。同时，物理变量的缩放比例作为整数运算中常见的误差源，在这里是不要求的。

然而，获得高精度的数值是以字长的增加为代价，反过来意味着，对于存储器容量和运算时间的要求越高。例如，在实时系统中，基于抢先占用的策略，调用和存储浮点数可能对运行时间有很大的影响。

因而，在许多应用中，把定点和浮点运算结合起来处理。因此，了解和理解常用的数值方法对解决问题是非常重要的。

（1）将定点整数转换为浮点数，反之亦然；

（2）处理除数为零的情况；

（3）由滤波器和整数算法产生的近似误差的传播；

（4）取整误差的传播。

比较和除法操作的要点如下：

（1）当不是必须进行定点数的比较操作时，在大多数情况下应该避免两个浮点数 a 和 b 的比较操作。而推荐对差值 $\delta = |a - b|$ 和 ε 进行比较，同时也要求考虑到相对准确性（如，以 $\delta < |a \cdot \varepsilon|$ 或 $\delta < |b \cdot \varepsilon|$ 的形式）。

（2）通过设定条件和查询，排除被零整除的情况。

5.4.2.11 建模和实现指南

ECU（电子控制单元）的最优化一方面信赖于应用，另一方面依赖于硬件平台。

因而，负责基于模型的物理规范的功能开发者和负责设计和实现的软件工程师之间必须密切协作。

建模和实施方针是进行专门最优化措施的重要前提条件。功能模型必须有助于对所有软件相关信息的直接说明，而无需出现对于理解软件功能来说不必要的信息。建模原则的例子有 MSR standards[79]，而 MISRA Guidelines 则是建立应用方针的例子[80]。

将需求分析和设计分开有利于转到新的硬件平台，这点很有必要。最好情形的假定中，这些可以通过针对具体硬件的设计决策的变化来实现。

需求分析和设计的连贯性是功能开发中的基本问题。许多数据和行为模型工具支持这些设计步骤。开发工具对于变量确定基本模型库、重新设定范围的推荐、变量的命名、数据结构、特性曲线和特性图的插补程序、存储器表等确定一些指南。

5.4.3 软件结构的设计和实现

在确定详细的软件结构时，必须考虑到指定的微处理器的特点和 ECU 的特性，同时考虑对 ECU 的所有要求。由于不同的限制条件，软件的结构只有在具体的应用中才能明确。

5.4.3.1　平台和应用软件

两个软件层的区别，如平台和应用软件，已经被广泛接受。在第 5.3 节为软件功能定义了一个体系结构。在设计阶段，详细说明了软件的组成，这些组成部分对于实现软件功能是必不可少的，这些组成部分可以集成为第 1 章和第 2 章中介绍的软件体系结构中的应用软件。图 5-66 所示为一个框架设计的实例，在这个例子中，软件功能以模块的形式实现，以信息的方式通信。该模型用图 5-24 中首次介绍的图形表示法描述。

接下来的部分讨论软件组件的实现方法和设置，尤其是那些适合于从恰当的工具中获得自动支持。以这种方式，能够确保需求分析和实现之间的一致性，对于软件质量的提高有至关重要的贡献。

5.4.3.2　平台软件组件的标准化

平台软件组成的标准化有一系列的好处。从汽车制造商的角度看，平台软件组件不提供竞争的可能，所以有可能实现标准化。平台软件组件的标准化，使得在机动车辆中对 ECU 的集成更为便利，这些 ECU 是由很多不同的卖主提供。而且，能保证平台软件组件的质量。例如，下面的平台软件组件已经实现了标准化。

（1）操作系统、通信和网络管理，依据 OSEK 标准[16]；

（2）诊断协议，依据 ISO 标准[25, 26]。

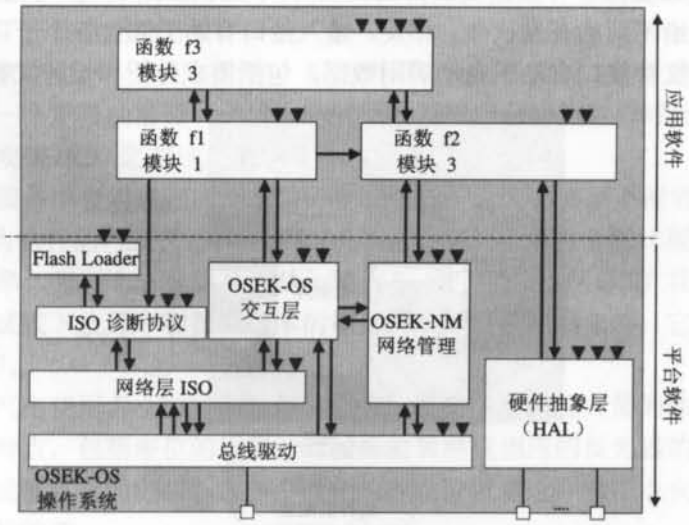


图 5-66　由标准软件组件构成的软件体系结构

在配置好参数以后，软件组成能够适合多种多样的应用。

闪存编程的标准化，包括必要的平台软件组件和必需的安全机制以防止侵权，也提供许多的好处。

在生产和服务阶段，软件组成的标准化支持为 ECU 提供一些功能的非车载接口：

（1）诊断；

（2）软件参数化；

（3）软件更新。

在开发过程中，常常需要提供附加接口，如用于测量和校准，或快速原型的应用。测量和校准协议，如 CAN 校准协议（简写为 CCP），或是 XCP 协议，这些都在 ASAM[17]的支持下实现了标准化。仅在软件开发阶段，要求对于软件框架中的功能必需的软件组件进行集成，而在生产和服务阶段并不再有此要求。

在应用软件的组件中，也有进行标准化的潜力，这一点是很显然的。如合适的选项将被插补到相应的程序或数据结构中，如特征曲线或特征图、或控制技术的系统库元素，该元素是在 MSR-MEGMA 工作组[79]的支持下的指定类型。

用来控制微控制器外围模块的软件模块通常被集成在硬件抽象层（HAL）中，对于给定的微控制器或微控制器簇，它们的应用可以被标准化。

5.4.3.3 标准化软件组件的配置

使用参数配置使得标准化软件组件能适应具体应用。配置步骤可以通过使用配置工具自动完成。关键部分是实时操作系统的配置或用以通信和诊断的平台软件组件的配置。

图 5-67 中表明了自动生成 ECU 网络内部通信软件组件配置设置的大概情况。为此，ECU 通信矩阵存储于一个中央数据库中，编辑的利用可使通信矩阵参数的修正更为容易。这样，就使得从信号角度、信息角度、总线角度、节点角度或功能角度进行特定的配置成为可能。输出接口支持多种数据交换格式（如开发或测量工具的描述性文件），通常用于把通信矩阵分发给不同的开发伙伴。相反，输入接口有助于综合各个子功能且来证实它们的一致性。一个文件接口有助于自动调用数据，包括需求分析和设计文档。

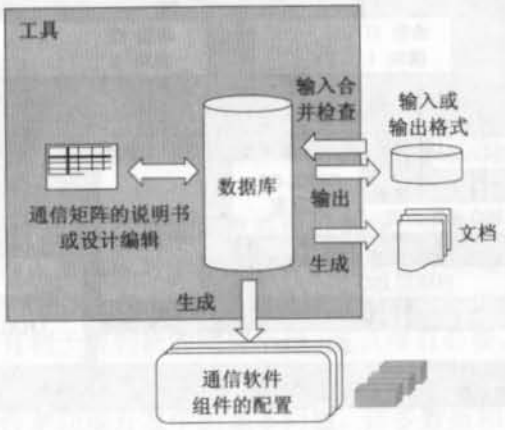


图 5-67 通信层的自动配置

以上所有的选项结合起来能确保实现、文档和 ECU 网络中通信数据格式描述的一致性。这样就可以避免在手工配置软件组件时产生传递误差。

在诊断数据中也有相似的要求（图 5-68），在一个中央数据库中管理诊断数据有很多好处，能自动配置以诊断为目的的软件组件，它确保了用于诊断测试仪的错误信息的存储记录（如 ASAM-MCD 2D 格式）与 ECU 车载实现之间的数据一致性。另一个可自动完成的任务则是将几个 ECU 中的诊断数据集成为一个独立的数据版本，并完成一致性检查。

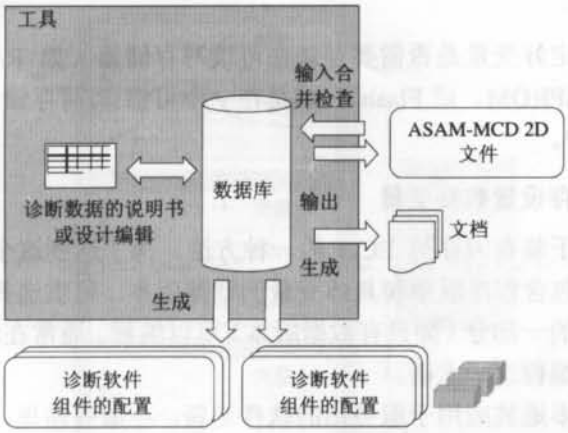


图 5-68　诊断层的自动配置和诊断说明书的产生

5.4.4　数据模型的设计与实现

不同的国家和消费者对于汽车配置的不同需求，导致在生产和服务中出现大量不同的汽车系列和型号。不同的汽车系列和型号导致汽车 ECU 的不同软件版本。

为了减少在生产和服务中必需的 ECU 类型的数量，这一节中，通过为 ECU 软件产生数据变量，引入了一些有用的方法。

（1）对于所有在生产中应用的程序，根据所需时间的长短为 ECU 建立程序或数据版本的需求构成了一个重要的前提。通过生产时间周期规定允许的最长时间，创建过程可以插在汽车中物理安装 ECU 之前或之后。

（2）在汽车服务中使用过程的情况下，全球性的后勤服务要求不同的 ECU 硬件类型的数目最少。作为软件的内在优点，程序和/或数据版本的全球性分配比硬件组件的分配需要更多的成本效率。另一个好处就是，从一辆汽车上拆卸和维护 ECU 花费很大。因此，有利于安装、调试或下载程序和数据版本的概念提供了很多有利条件，它们并不需要拆卸 ECU 或交换 ECU。

（3）而且，汽车使用者（如操作者和其他人）更希望能通过大量的软件功能可以配置和存储个性化的特点，包括座位的设置、驾驶杆的调整或相应的反光镜的位置，加热系统或空调的设定，或最喜欢的电台。这些个性化的外设可以通过存储在点火装置钥匙中的驾驶员身份信息进行管理。

在数据设计和应用的过程中必须考虑上面提到的所有需要考虑的事项。

5.4.4.2 和 5.4.4.3 这两节中详细地介绍了数据版本的安装或配置的两种不同方法：

（1）通过闪存设置数据变量；

（2）通过参数配置设置数据变量。

另外，也可以把这两种方法结合起来。

5.4.4.1　存储单元的定义

除了微处理器中使用的描述类型外，对于每段数据，必须确定数据存储在微控制器的

那个存储单元。

相应地，必须决定好变量是否需要存储在可读写存储器（如 RAM）或在只读存储器（如 ROM、PROM、EPROM，或 Flash），或是在一个可擦读/写存储器（如在 EEPROM 或电池支持的 RAM）中。

5.4.4.2 通过闪存设置数据变量

第一种方案是基于装有闪存的 ECU 的一种方法。为了达到这个目的，在整个闪存区域中可以编程，其中包含程序版本和具体变量的数据版本。可供选择的是，当汽车下生产线时也可以只有闪存的一部分（如只有数据版本）可以编程。通常在汽车生产结束时编程，因而这就产生了线尾编程这一术语。

同样的方法也更多地被应用于服务站的软件更新，在服务站里，通过汽车的中央诊断接口来编程，为此，ECU 并不需要从汽车中拆卸下来。服务站里的闪存编程将在本书的第 6.3 节详细讨论。

为了减少闪存编程的时间，程序版本和数据版本常常是分开进行。在生产情形下，如在 ECU 制造的过程中，与汽车型号系列无关的程序版本已经被编程，而只有与具体的汽车型号系列相关的数据版本属于线尾编程。

以特性曲线为例，图 5-69 所示为一个通过闪存编程的变量管理的例子。

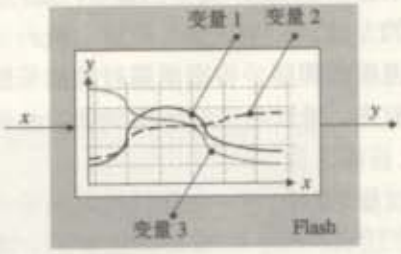


图 5-69 以特性曲线为例的数据版本闪存编程

5.4.4.3 通过配置参数设置数据变量

第二个解决方案是在 ECU 板上只读存储器中不同的数据变量并行存储。这些数据变量中只有一个被选择用于进行随后的线尾参数化。相应的软件参数可以存储在 EEPROM 上。因为只有当汽车即将离开生产线时，才选择几个可能的配置中的一个（如第 3 章中图 3-12 所示），这种方法被称做线尾参数配置。

一个可选用的方法是，在汽车起动时，可以选择各自的配置。在这种情况下，描述程序将所有的车载 ECU 车辆的配置数据存储在一个中央 ECU 中。从那里，例如，在点火之后，数据将在网络上以信息的方式发送给接收器，一旦收到信息就选择所必需的配置。

这种方法也可应用于野外汽车中的软件配置。这里，中央诊断接口也可以用做连接点。另外，这种方法便于汽车操作者自己配置个性化的功能参数。

以特性曲线为例，图 5-70 所示即为一个通过软件参数实现配置管理的例子。

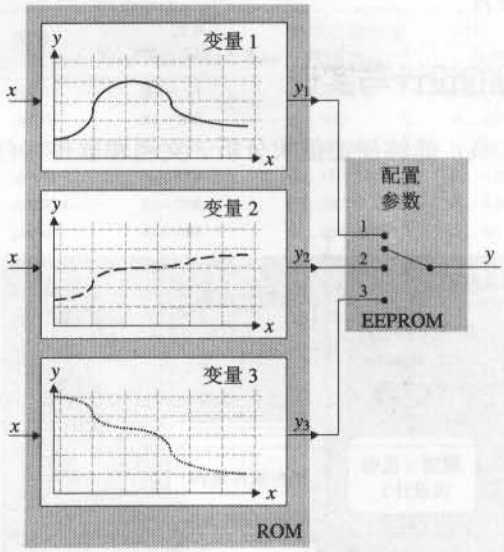


图 5-70　以特性曲线为例的数据版本参数化

5.4.4.4　数据结构和描述文件的生成

通过数据结构以及用于测量以及数据校准的描述文档的集中管理和自动生成，有利于下面步骤的自动化（图 5-71）。

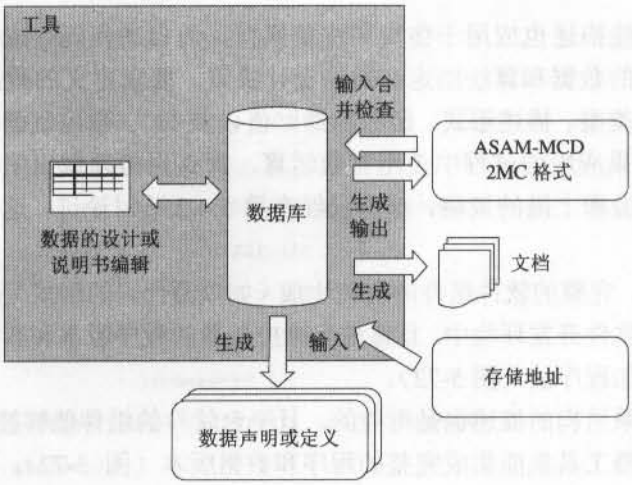


图 5-71　数据结构和描述文件的自动生成[89]

微控制器中的测量数据和校准数据存储在中央数据库中。在这个过程中，所有数据的物理规格、设计和实现决策，以及转换规则的定义（如通过使用变换公式），可以一起处理。一方面，数据库中的记录可以用于在一个开发环境（如在 C 语言中）中产生数据结构（见第 4 章图 4-35）。另一方面，一旦地址信息被装载，则生成 ASAM-MCD 2MC 格式的描述文档所必需的所有信息都是可用的。这些描述文档可以应用于测量、校准以及诊断的工具中。这样，能够保证用于测量、校准工具的描述文件以及 ECU 板上的微控制器中的

实现三者之间的数据一致性。

5.4.5 行为模型的设计与实现

通过使用代码生成工具，能够保证需求分析、文档和软件组件的实现之间的数据一致性（图 5-72）。

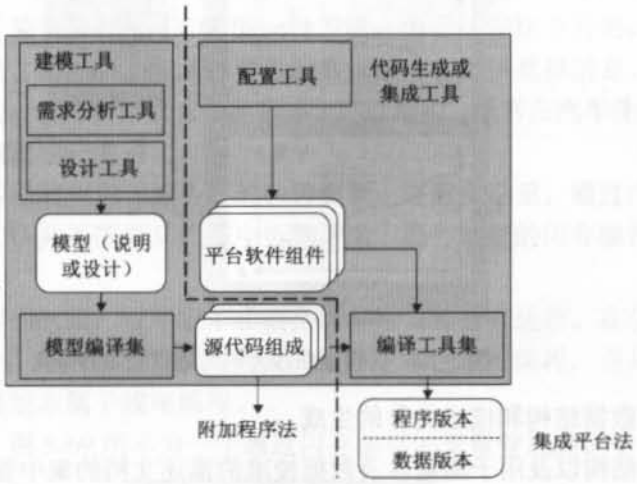


图 5-72 软件组件、程序和数据版本的自动生成

基于模型的功能描述也应用于仿真和快速原型，为自动代码生成提供依据。这意味着，必须为物理层的数据和算法制定必需的设计决策。要求定义的数据如图 5-71 所示，算法要求为变量的类型、描述形式、函数的返回值以及 ECU 微控制器的存储单元的分配做出设计决策。如果应用的过程中使用整数运算，那么也必须做出附加定义，如关于取整误差的处理、下溢和上溢的策略，这些已经在第 5.4.2 节讨论过，这些定义可以通过设计工具得到。

根据这些信息，完整的软件组件的自动生成（如以源代码的形式）是有可能的。随后，这些可以应用常规软件开发环境中，且集成到微控制器的程序版本和数据版本中。相应地，这个方法被称为附加程序法（图 5-72）。

如果软件的体系结构的描述也是可能的，且平台软件的组件能够被集成和配置，因而通过处理器的编译器工具集能集成完整的程序和数据版本（图 5-72）。相应地，这个方法即为大家所熟知的集成平台法。

举例：积分器类的设计和应用

目标：实现图 5-27 中介绍的积分器类。因而，对于数据做出如图 5-73 的设计方案。

如图 5-74 所示，用 C 语言和定点算术来实现“compute()”方法的一种可能的实现方式。算法中，还考虑其他的设计决策，如溢出或下溢的处理。

指定 X	符号	公式 $X_{impl}(X_{phys})=K_1 * X_{phys}$	物理层的取值范围 X_{phys}	应用层的取值范围 X_{phys}
E/compute()	uint8	$K_1=$ 1	true/false	1/0
in/compute()	uint16	$K_1=256$	0 ... 100	0 ... 25 600
K/compute()	uint16	$K_1=256$	0 ... 255.996	0 ... 65 535
MN/compute()	uint32	$K_1=256$	0 ... 16 777 215.99	0 ... 4 294 967 295
MX/compute()	uint32	$K_1=256$	0 ... 16 777 215.99	0 ... 4 294 967 295
return/out()	uint16	$K_1=256$	0 ... 255.996	0 ... 65 535
I/init()	uint8	$K_1=$ 1	true/false	1/0
IV/init()	uint16	$K_1=256$	0 ... 100	0 ... 25 600
memory	uint32	$K_1=256$	0 ... 16 777 215.99	0 ... 4 294 967 295
dT	uint16	$K_1=1024$	0 ... 63.999	0 ... 65 535

图 5-73　积分器类中的数据和接口的设计方案

```
/* Variables */
extern uint32 memory;
extern uint16 dT;

/* Method compute() */
void compute (uint16 in, uint16 K, uint32 MN, uint32 MX,
uint8 E)
{
  uint32 t1uint32, t2uint32, t3uint32;
  if E {
    /* Overflow handling 15 Bits */
    t1uint32 = MX >> 1;
    /* min=0, max=2147483647, impl=128phys */
    t2uint32 = (( (uint32) (in >> 5)
                 *( ( (uint32) K * dT) >> 10)
                ) >> 4)
             + (memory >> 1);
    /* min=0, max=2357192447, impl=128phys+0 */
    t3uint32 =
(uint32)((t2uint32<t1uint32)?t2uint32:t1uint32)<<1;
    /* min=0, max=4294967294, impl=256phys+0 */
    memory = (t3uint32 > MN) ? t3uint32 : MN;
    /* min=0, max=4294967295, impl=256phys+0 */
  }
}
```

图 5-74　作为 C 语言的一个函数，积分类中“compute()”方法的实现

5.5 软件功能的集成和测试

这一节讨论在集成和测试阶段，用于软件功能的有效性和正确性验证方法。由于集成和测试过程通常跨越多个企业间进行，因而用于建模和仿真技术的那些方法在汽车开发过程中起关键作用。这些方法能够把不存在的物理组件用虚拟组件来代替。

本节主要围绕下面的集成和测试环境展开：

（1）仿真工具；

（2）实验室汽车和试验工作台；

（3）实验的、原型的、真实的汽车。

在所有开发团队和开发环境之间，各种各样的有效性和正确性验证步骤的同步，如组件模块和测试案例的协调，必须在项目计划中尽早考虑到。

在 5.3 节已经讨论过一些软件功能的有效性验证方法，如具体软件功能的快速原型，应该在需求分析阶段做好。前面已经讨论过的方法能够与本节要研究的方法相结合。这里强调的方法是，在半虚半实的环境中，尽早地实现已有功能的有效性和正确性验证。

为此，引出几种典型中间步骤。图 5-75 中描述的是，开始的情形和最后目标。

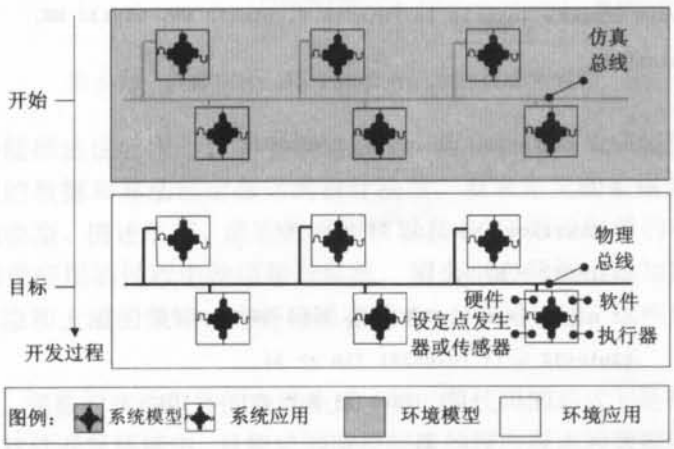


图 5-75 软件与系统集成和测试的初始状况和目标

图 5-76 介绍了在集成阶段各种各样的中间步骤的总体状况。表示逻辑系统结构的模型，可以为不存在的系统组件的仿真提供依据。以下部分基于从图 5-76 选择的样品，讨论被广泛认可的集成方法，以及软件功能的有效性和正确性的检验方法。

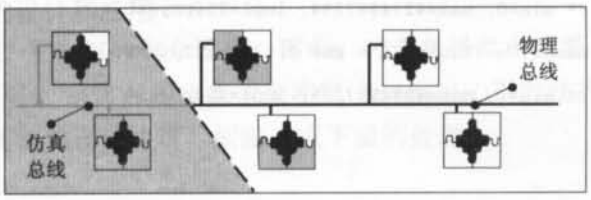


图 5-76 软件与系统的集成和测试的中间步骤

最早可能的检验步骤包括控制函数模型与被控系统模型的仿真，仿真模型可以模拟，如设定点发生器、控制器或监视器、调节器、被控对象和传感器等部件。对于许多汽车功能，仿真也能研究车辆操作员以及环境对系统行为的影响。操作员和环境也许被作为仿真的组件，如图 5-44 所示。

由于虚拟的模型表示了所有的组件，所以对于建模过程或仿真过程的实时性要求是不存在的。关于汽车组件的建模的详细讨论已经超出了本书所要研究的范围。关于功能建模的详细情况，可见参考文献[35]。

5.5.1　回路软件仿真

如果已经实现的软件组件在仿真环境中运行，这种方法称做回路软件仿真（简称为 SiL 仿真）。如果它表示一个闭环控制功能，那么这个称谓有意义。例如，一个执行闭环控制功能的应用软件层的软件组件可以作为回路中的组件被模拟和执行，如图 5-77 所示。

然而，即使不存在这样的控制回路，该方法也可以被很好地利用，并且有大量相关的其他应用例子。例如，软件组件过去常常用于开环控制或监视功能的情形，或者平台软件的软件组件（如通信层），就是以这种方式被核实和确认的。

在这种情况下，仿真模型的结构与图 5-44 中所示的没有很大的差别。然而，却需要对组件的模型进行相当具体地确定。例如，ECU 的建模过程必须相当具体进行模/数和数/模信号转换，同时实时行为都必须尽可能地准确考虑。只有这样，在这个系统环境中集成并运行已实现的软件组件才是可能的，如图 5-77 所示。被执行的软件组件，这时已经变成一个测试品，在仿真环境下的开发和仿真平台上（例如在一个 PC 机上）被运行。这里没有关于仿真执行的实时要求。

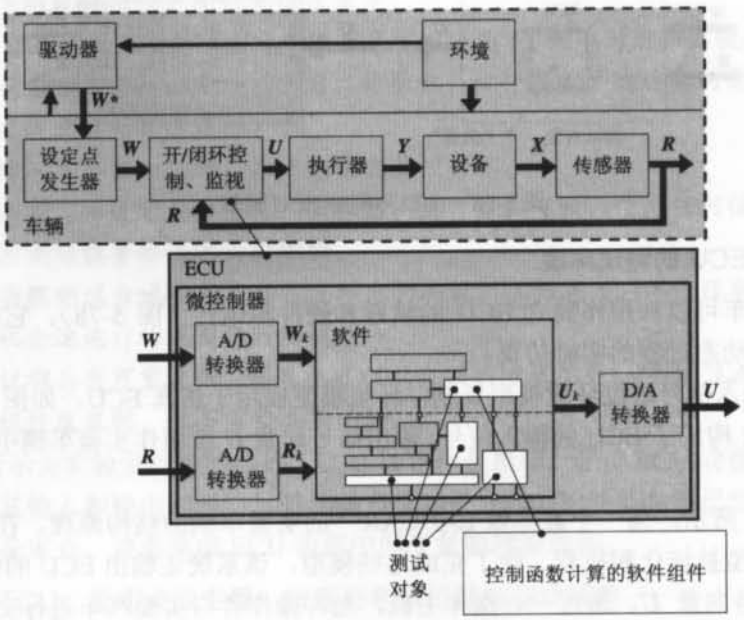


图 5-77　回路软件仿真

回路软件仿真的应用便于大量动态软件在早期阶段、在没有真实的控制器的情况下运行，例如，带有代码覆盖分析的组件测试。

5.5.2 实验室汽车和试验台

当一个给定的 ECU 当中的硬件和软件可以利用时，应用于有效性和正确性验证的一大类的方法和工具被统称为实验室汽车和试验台。目标是在一个半虚半实的环境里，执行真实 ECU 的运行，如图 5-78。因此，与前面提到的仿真方法相比较，在仿真环境组成的建模和执行中必须考虑实时性要求。

如果控制功能的核查和验证是重点，那么 ECU 必须当作控制回路中的一个组件来考虑和对待，如图 5-78 所示。为此，这种做法就是通常所说的硬件回路仿真（简称为 HiL 仿真）。与 SiL 仿真类似，这种方法不仅仅限于控制功能，也能被用于大量的其他应用中，下面将讨论一些例子。这些不同方法在本书中统称为实验室汽车。

ECU 软件的许多方面，如核查和验证实时行为，网络上的车载和非车载通信行为，或控制和监控功能的验证，都是至关重要的。

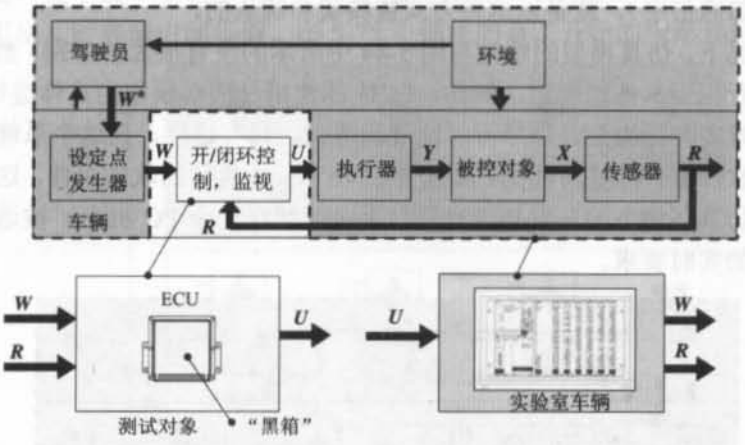


图 5-78 虚拟环境中 ECU 的运行

5.5.2.1 ECU 的测试环境

实验室汽车可以被用作独立 ECU 的软件和硬件测试台（图 5-78）。它能够生成 ECU 环境的静态和动态过程的实时仿真。

ECU 的输入信号通过环境模型复制，环境模型被用于仿真 ECU。如图 5-78 所示，信号向量 ***W*** 和 ***R*** 构成了 ECU 的输入信号。输出信号向量 ***U*** 被用作实验车辆中环境仿真的输入变量。

如图 5-79 所示，为一个被称做 LABCAR[90]的实验车辆的结构原理。在计算机实时系统中，环境模型被转化和执行。除了完成这些模型，该系统还输出 ECU 的信号向量 ***W*** 和 ***R***，且收到信号向量 ***U***。通过一台操作主机，允许操作者与实验汽车进行交互式控制，或者实验汽车的自动控制。建模工具支持环境组件模型的修改。

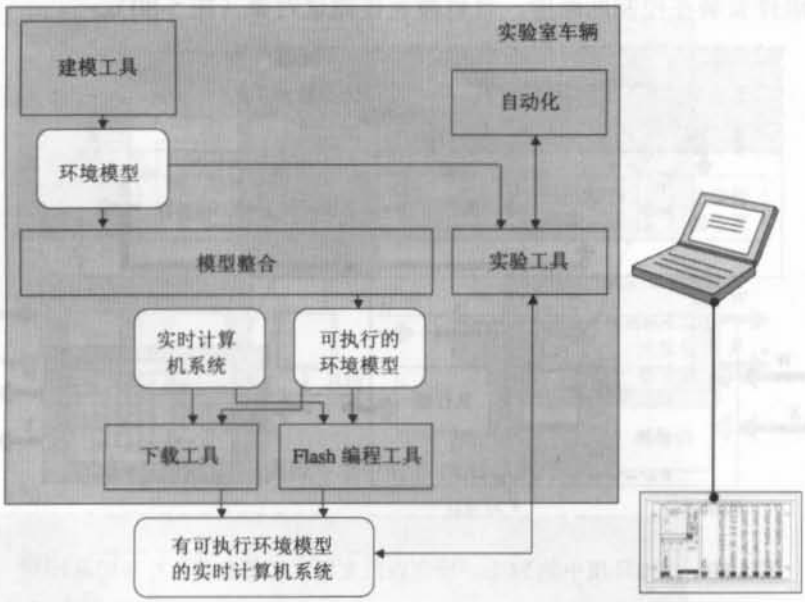


图 5-79　LABCAR 实验车辆结构图

举例：ECU 控制功能的测试环境

实验车辆的典型应用，包含对 ECU 提供的控制功能的动态特征的测试。在应用软件中，除了软件功能，还包括软件平台和 ECU 硬件的信号处理。

ECU 输入信号预调选项的自由性，使得实验测试的进行具有很大的自由度。

（1）来确定 ECU 的环境条件，如温度、大气压力或湿度。通过输入信号的随机激励，使得极端情况下的软件功能的测试变得很简单。

（2）这样，在对测试人员和原型汽车没有危害的情况下，能够对边界驾驶情形进行仿真。

（3）也可以随意指定如设定点发生器、传感器、执行器或导线连接的老化或故障的情况。可以对于自适应控制功能进行评价。

（4）通过输入不真实的信号，系统地检查监控功能。

（5）组件公差，如设定点发生器、传感器、执行器，可被预设为任何值。这样，可以测试它们对于控制功能鲁棒性方面的影响。

（6）对比物理测试台试验或在真实汽车中的测试，工作点可以不受限制地随意给定，如可以在发动机全速运行的范围进行测试。

所有的测试都具有可重复性，并且也可以以自动的方式运行。对于“LABCAR”测试，没有其他硬件组件或装配，也不需要真实车辆。

图 5-78 所示为实验室汽车的结构，ECU 被看作是黑箱。评价 ECU 功能行为的唯一方法，就是分析其输入和输出信号。尽管这种方法对于 ECU 的简单功能已经足够了，更多复杂功能的繁杂审查，要求集成 ECU 内部中间变量的测量过程。

5.5.2.2　ECU、设定点发生器、传感器和执行器的测试环境

前节所述的方法还可扩大到包括设定点发生器、传感器和执行器。为了实现这些功能，

这些真实的组件安装在控制回路中，且也被看作测试对象（图 5-80）。

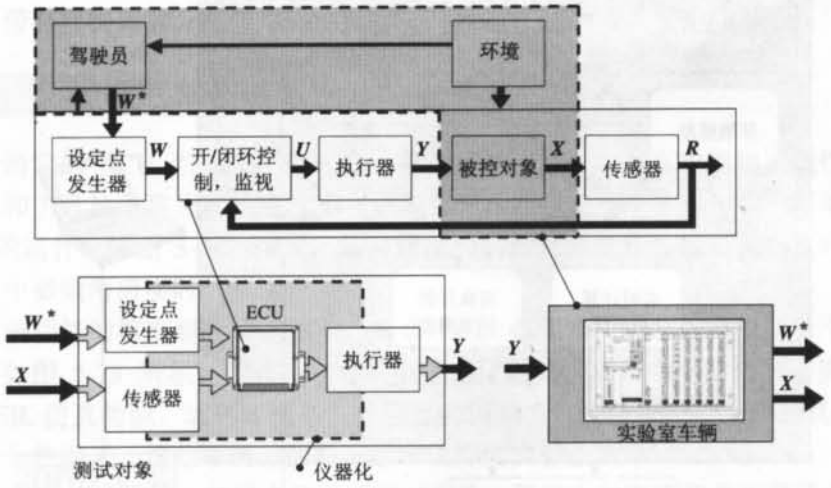


图 5-80 虚拟环境中的 ECU、设定点发生器、传感器和执行器的运行[90]

这样，在实验车辆中，建模仅限于建立被控对象，车辆操作员的和环境的模型。不需要再建立表示设定点发生器、传感器、执行器的模型。在这种特殊的情况下，就要求实验车辆支持输出变量 ***W**** 与 ***X***，和输入变量 ***Y***。这反过来需要对硬件结构进行适当的修改。

同样，模拟和真实的设定点发生器、传感器和执行器的结合使用也是可能的。

举例：控制和监控系统的测试环境

与前面例子的特定目的相比，这个设备为电子控制和监控系统提供综合测试。测试设定点发生器、传感器和执行器要求测试在这些组件中产生的不同的信号。在大多数情况下，这些信号被另外的一组传感器获得。获得 ECU 内部变量的过程被称做 ECU 的仪器使用。

组件和 ECU 的仪器使用，可以与测量、校准、诊断系统一起，集成在实验车辆中。ECU 可以通过几个非车载接口中的一个进行存取，如非车载诊断界面。

该测试案例在某些地方上超过了前面例子的测试情况。但是，在其他地方也存在着限制，例如传感器上老化效果的预置，或对极端情况和错误事件的设置等。

5.5.2.3 ECU 网络的测试环境

如果被测试的功能是以分布式系统和网络系统的形式实现的，这表明必须扩展测试过程来处理几个 ECU 的同步测试。也就是说，这些仪器必须扩展到包含几个 ECU（图 5-81）。

测试常常分阶段进行。例如，第一个阶段集中在测试 ECU 总线通信和软件平台的相关组件上，第二个阶段是测试应用软件组件。

在这两种情况下，不存在的 ECU 可以由它们的相应的仿真部件代替，并且成为计算机实时系统中环境模型的组成部分。为此，计算机实时系统必须提供一个总线接口，以便组成通信系统。

描述 ECU 通信行为的虚拟仿真，且与已经实现的部分网络系统功能互联，以测试 ECU 网络上的通信，称做剩余总线仿真，如图 5-82 所示为其中的一个应用例子。

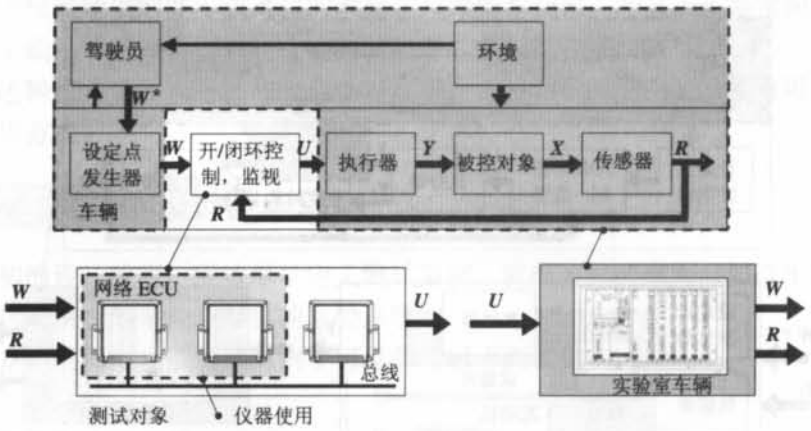


图 5-81　虚拟环境中的多 ECU 运行[90]

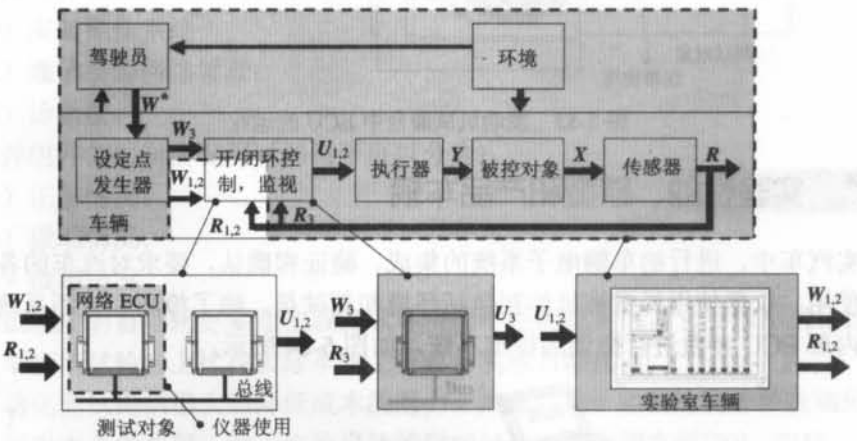


图 5-82　虚拟环境中真实和虚拟 ECU 的运行[90, 91]

5.5.2.4　测试台

从实验车辆到测试台的转换是非常自然的。如果电信号不足以驱动执行器，例如电液执行器，那么一个适合的测试环境就是液压测试台。

与在实验室汽车上工作相比，整个系统更多的是真实部件，例如被控对象（如图 5-83 所示）作为测试对象，而集成在测试环境中。此外，也可以重新定义环境状态变量（例如测试设备的环境温度可能极冷或极热。类似的选择可以通过驾驶员预先设置期望值来完成。

若真实的组成部件还不存在，则以虚拟的形式复制，比如以驾驶员或者环境模型的方式，通过它们可以调整动态负载情况。这些虚拟部件可以作为实验车辆和试验台的标准模块使用。一个计算机实时系统（如一个实验车辆）集成在试验台上，以便完成仿真。

举例：发动机试验台

图 5-83 描述的发动机试验台中，把 ECU、设定点发生器、传感器、执行器和发动机看作是试验对象。其余的汽车组件、环境条件、工作状况都是虚拟的。

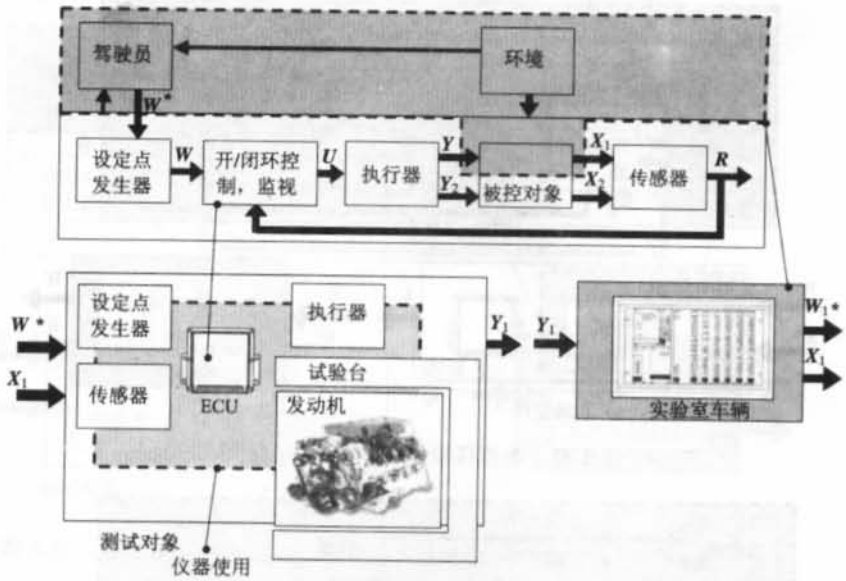


图 5-83 发动机试验台中 ECU 的运行

5.5.3 实验性的、原型和产品车辆

在真实汽车中，进行的车辆电子系统的集成、验证和确认，要求对汽车的各个组成部件的仪器使用。仪器使用经常被延伸到包括环境和驾驶员。除了检测和诊断系统外，还需要一个对内部 ECU 参数进行微调的校准系统，如图 5-84 所示。

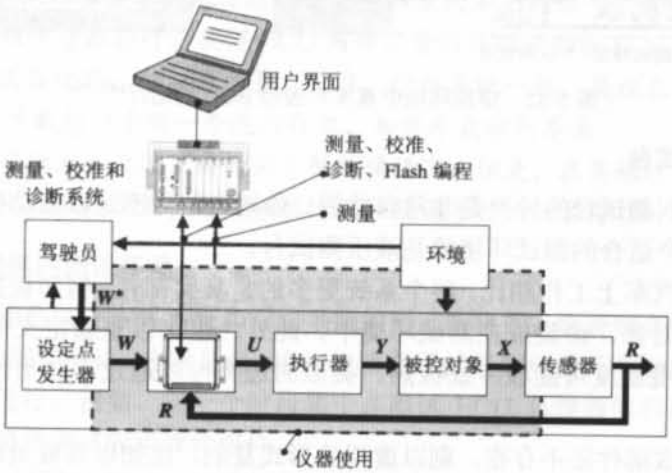


图 5-84 试验车辆中的仪器使用[86]

按理说，测量、校准和诊断系统必须适合在网络上分布的 ECU 功能的集成。为此，它们必须支持多个 ECU 的同步仪器使用和标定。测量和校准系统将在第 5.6 节进行详细地讨论。

这种测量、标定和诊断系统也可以与快速原型系统相结合，如图 5-47 所示。

在许多情况下，从原型到产品汽车的转换，以及从开发 ECU 到产品 ECU 的变化，也

导致了用于仪器使用的非车载接口的改变。与原型车相比，产品汽车常常限制 ECU 存取主要的非车载诊断接口，而且，在某些情况下，限制存取通信系统。

而上述转换导致非车载接口的传输速度减小，限制了校准功能的参数可选择性，而且改变了工作方式。

5.5.4　实验的设计及自动化

在最初的设计阶段，应该尽早定义测试案例。依据不同的准则，如汽车功能、汽车系统组件或驾驶情况等来构造具体的实验系统。

软件功能的面向功能测试的例子可以作为以下测试案例：

（1）控制功能；

（2）监控和诊断功能。

面向组件和系统测试的例子可以作为软件组件的测试案例，例如：

（1）实时操作系统；

（2）通信层和网络管理；

（3）诊断层。

软件组件的面向情况测试的例子可以分为：

（1）正常情况；

（2）极端情况；

（3）故障情况。

测试程序的自动化更多地依赖测试环境而不是测试案例，它要求形式化地描述实验。在实验车辆或试验台上比在实际车辆上更容易实现自动化。

自动化测试提供很大的降低成本的潜力。然而，关于实验的设计和自动化操作的详细讨论已超出本书的范围。对于实验设计的详细讨论，请参阅文献[30]、[31]。

5.6　软件功能的校准

在实验车辆、测试台以及真实汽车中，测量和校准系统的各种可能的应用，在第 5.5 节已经讲过。本节讨论测试和校准系统的功能原理。

测量和校准系统包括一个测量和校准工具、一个或多个 ECU，每一个 ECU 是以一个或多个特殊的微控制器为特征的，这些微控制器上配有合适的非车载接口。另外，还有捆绑于所使用仪器之内的辅助测量技术。

在工具中，通过使用仪器使用方式获得的所有信号，必须用标准的模式表示。这不仅适用于值域，也适用于包括已被测量信号的时间域限制。对于微控制器程序获取的离散测量信号，这意味着从用于实现层的符号到用于测量工具的物理表示的变换是必需的。

对于参数值的任何变化，如特征值、曲线和图的值，需要这种工具提供编辑在实现层和物理层的支持。图 5-85 所示为一个从物理层角度和实现层角度的特征曲线 KL 和测量信号 S 的例子。

因此，区分由 ECU 微控制器完成的车载工作步骤和那些通过测量、校准工具完成的

非车载工作步骤是非常有帮助的（图 5-85）。

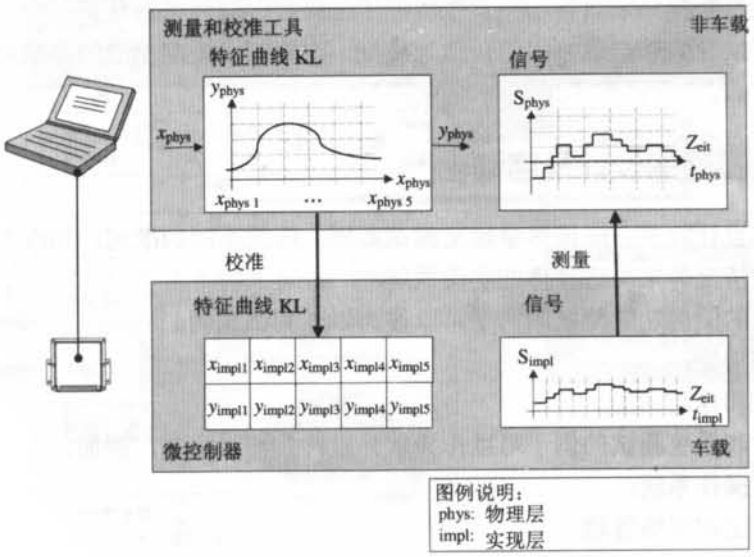


图 5-85 测量和校准系统的车载和非车载计算

开发阶段的目标是生成或改写数据版本，这些数据版本中包含以特征值、曲线和图的形式存储在微控制器存储器中的所有参数值。微控制器的程序是以这些数据作为操作的基础。

为了完成那些通过分布式和网络系统的方式实现的软件功能的校准，测量和校准系统必须支持完整的微控制器与 ECU 网络。为清晰起见，下面应用最简单的方法，其中仅仅涉及一个微控制器和一个 ECU。

一开始就要准备一个 ECU，也就是一个硬件和软件版本。软件版本包含一个程序版本和每个车载 ECU 上的微控制器的初始数据版本。测量和校准系统还需要有一个软件版本的描述，它可能以 ASAM MCD2 格式的文件形式被存储。该文件不仅包括用于所有的测量、校准和诊断数据的物理层和实现层之间转换的信息，还提供关于工具和微控制器之间的接口的信息。

有时，要求近乎苛求的以及花费巨大的开发步骤的目的是为了便于改写数据版本。在众多因素中，有几个方面是主要的。其中有对不同工作点的适应；在系统的长期运行中，以参数和算法的方式修正老化的影响；快速试验便于评价组件制造误差，或者不同类型、型号汽车数据版本的改写。

5.6.1 离线和在线校准过程

当校准系统工作时，离线和在线校准之间有着根本的区别。

在离线校准中，当参数值被修正或校准时，ECU 的控制和监视功能的执行（所谓驱动程序）是断续的。因此，离线校准受到了很多限制。尤其是用于测试台或车内测试时，设置或改变参数总是要求中断测试台或驾驶试验操作。

为此，支持在线校准的更多通用方法的测试程序更为有用。在线校准中，参数值可以在运行时随时修改，即车载 ECU 上的微控制器正在运行驱动程序时也可以修改。换句话

说，参数的设置或修改，与控制和监控功能的同步执行是可能的，甚至是在正常测试台和驾驶试验操作正在进行的时候也是可能的。

在运用校准工具的过程中，例外的情况可能出现。例如当特征曲线和图沿着 x 轴上的点严格单调递增时会出现短期的故障，这些短期故障要求微控制器中运行的程序有一定的鲁棒性。因此，在线校准对于控制和监控功能的稳定性有更高的要求。

对于某些具有较低动态特性的功能参数，在线校准适合较长的反复迭代的参数协调，如在发动机试验台上的微调发动机控制功能。

在标定带有高动态控制功能的参数或安全相关的参数时，尽管参数不是在控制功能的运行期间进行调整，但是此时，也可以通过在线校正的方式，以反复迭代的方法来进行，避免了运行程序的中断。

一个相关的例子是在制动应用方面的 ABS 功能的微调。在这种情况下，尽管在实际的 ABS 控制行驶过程中不会发生调整，但是在线校准可以减小两次道路测试之间的时间间隔。在线和离线校准过程的两种工作方式对比说明如图 5-86 所示。

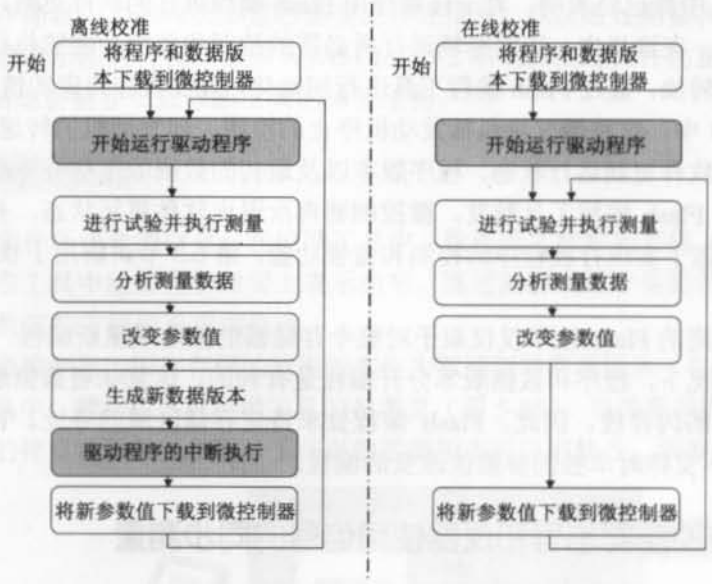


图 5-86　在线和离线校准程序对比的两个例子

图 5-86 概括了在线和离线校准系统中存在的不同要求。例如，离线校准系统，通过 Flash 编程的方式完成测量、非车载参数校准以及程序和数据版本的下载工作。相比之下，在线校准需要另外功能，这些功能有利于实时校准，且不需要中断行驶程序。下面的部分主要围绕在线和离线校准过程所需的功能这一主线进行讲解。

5.6.2　通过 Flash 编程的软件更新

要启动 ECU，首先必须将程序版本和数据版本下载到微控制器的相应存储器中。开发用的 ECU 通常装备有闪存。程序和数据版本的软件更新可以通过 Flash 编程完成（图 5-87）。

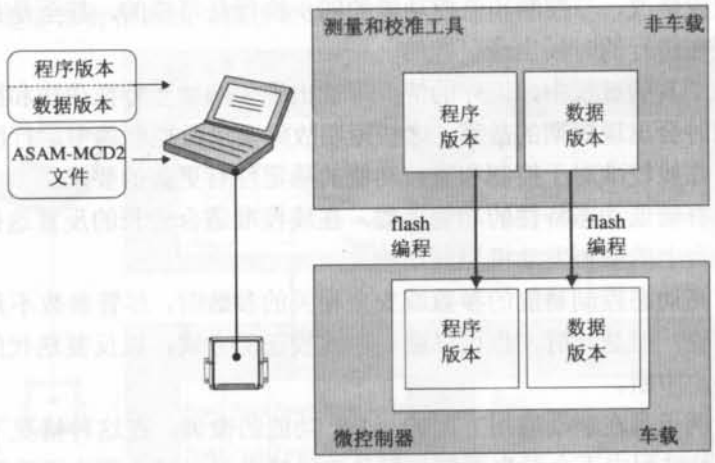


图 5-87 程序和数据版本的 Flash 编程

早在表 4-23 中就已经表明，对于任意使用 Flash 编程进行的软件更新，定义一种专用的软件运行状态，在这当中，正常车辆运行所必需的控制和监控功能的执行被中断。软件更新操作状态的转换，通过 Flash 编程工具进行初始化，且仅仅在特定的情况下才会发生。如在发动机 ECU 中，这种情况将包括发动机停止的检测，即发动机的转速为 0。

一旦进入了软件更新运行状态，程序版本以及最初的数据版本就被下载到微控制器的闪存。之后，由 Flash 编程工具触发，微控制器再次退出软件更新状态，并且转换到正常的状态，在该状态下实现行驶程序的控制和监视功能。第 6.3 节讲解用于软件更新的 Flash 编程过程。

在当前所使用的 Flash 技术仅仅限于对整个存储器的擦除或重新编程，即所谓的 Flash 段。在必要的情况下，程序和数据版本分开编程是有利的，这意味着数据版本必须是与程序版本存入不同的闪存段。因此，Flash 编程要求特定存储区域的寻址。目前的技术条件下，Flash 技术不支持对单独的参数值改变的编程。

5.6.3 微控制器信号和仪器使用信号的同步测量

通过测量可以评估参数值的改变所产生的影响。目的是在通过多种信号的测量，在执行特定的车辆功能时，分析所有车辆系统组件之间的相互协调作用。这种试验报告的一个典型的例子，就是在冷起动条件下，检查发动机 ECU 中对于氧气的控制。这个试验可以在零度以下的测试设备中进行，也可以在寒冷的试验场通过车内测试完成。

然而，在多数情况下，如果没有所有参与的车辆组件的仪器使用并使用恰当的测量技术，在微控制器和仪器组件上，数据同步的测量和记录，及用这种方法来评价一种功能几乎是不可能的。

测量系统必须支持微控制器中波动信号的捕获，比如车载微控制器上的 RAM 中存储的变量的测量值。这些测量也必须与 ECU 外围的仪器产生的附加信号同步（图 5-84）。最后，这些测量也可以包含与驾驶员或环境有关的信号的获取。

在与被测车辆功能相关的传感器和驱动器中，这些附加信号被频繁地获取。在许多情况下，当前的环境条件（如大气压力和空气温度）以及表示转矩、压力、温度或废气排放，所有的这些都在车辆不同的测量点被获取。同样，与车辆功能相关的数据通信可以从车辆的车载通信网络中获取，在许多情况下，并被同步记录。

因此，用于仪器使用的传感器的性能要求比车辆使用的传感器的性能要求要高。从测量的技术角度，在两个领域中必须接受的主要挑战是：一是微控制器信号的多种捕获率的同步任务；另一个是空间分布的和分散仪器的测量值的获取。

5.6.4　下载和评估车载诊断数据

除设置控制功能的参数值之外，监视和车载诊断功能的参数的校准也是必需的（如可行性检测的阈值的校准）。除了讨论的测量技术之外，车载诊断系统功能的实验验证也要求从微控制器的故障存储器下载并评价诊断数据。为准备实验也可能要求清除故障存储器。这表明，在校准过程中，已经要求具备非车载诊断系统的基本功能（见第 2 章图 2-64）。

用于故障存储器内容的纯文本显示需要的描述信息，以及通过测量和标定工具的信号从实现层到物理表示法的转换，是以 ASAM-MCD 2 格式的描述文件的重要组成部分。第 6 章中将较详细地讲解非车载诊断工具的体系结构。

5.6.5　离线校准

利用描述文件所包含的信息，在校准工具中，数据版本参数的数值（即特征值、曲线和图表）在校准工具中能够在物理层上表示出来。通过图解或基于桌面的编辑功能，这个工具也支持参数值的方便而直观的修改。

在下面，微控制器的闪存存储的数据版本作为微控制器参考版本（也称做参考页）。同样，在标定工具中，数据版本被称做工具的参考页（图 5-88）。要改变参数值，在校准工具中产生参考页的拷贝被称做工作页。工作页的数据版本可以被修改，而参考页保持不变。

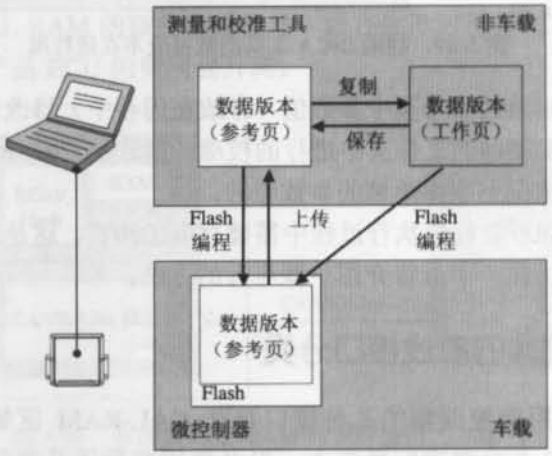


图 5-88　利用 INCA 工具的数据版本离线校准

工具的参考页可以支持工作页的变化，这样，工具参考页就成为与其他的工作页的改变进行比较的基础。

相反地，一旦当前值被加载到微控制器，工具的参考页中就包含了 ECU 当前状况的一个真实反射。

工具的参考页或工作页，仅仅同一个随后的 Flash 程序段被下载到微控制器，这将再次要求改变微控制器运行状态，同时要求运行程序的中断。

5.6.6 在线校准

如果要进行参数值在线修改，微控制器上需要扩展工作页或参考页概念。

这些通过把在线校准数据从 ROM 或闪存拷贝到（未使用的）RAM 区（即工作页所在的区）（图 5-89）来完成。在这个 RAM 区，也叫校准 RAM（简称 CAL-RAM），微控制器和校准工具中的校准数据需要同步（图 5-1 和图 5-89）。

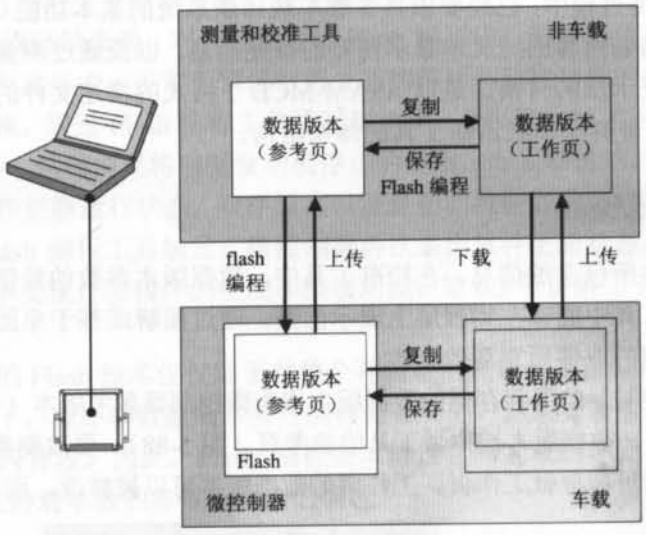


图 5-89 利用 INCA 工具的数据版本在线校准

因此，尽管微控制器的参考页中参数值（存放在闪存中）修改需要准确的存储地址和行驶程序的中断，微控制器的工作页中进行的校准（存放在 CAL-RAM 中）是能在无需影响或中断行驶程序的情况下获得单独的参数值的。

总之，微控制器的行驶程序执行过程中需要存取工作页，这是通过对微控制器软件或硬件的修改实现的。为此，下节将介绍一些可行的方法。

5.6.7 在线校准的离线接口分类

校准工具可以利用微控制器的几种接口存取 CAL-RAM 区域。基本的分类方法是根据 CAL-RAM 段是否在微控制器板上，以及使用并行还是串行方式与工具软件通信（图 5-90）。

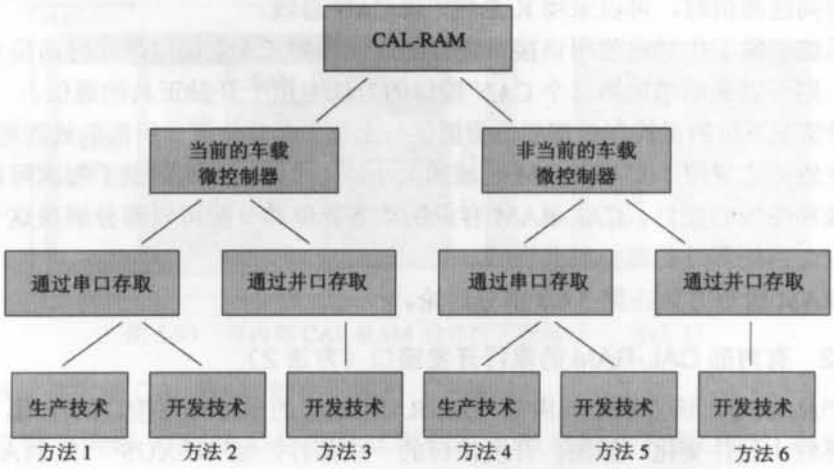


图 5-90　微控制器和校准工具间的接口分类

在串行通信方面，广泛应用串行通信方式，如 K-Line[5]或 CAN(Controller Area Network)[2]。仅在开发阶段用它进行软件下载和调试，如 NEXUS[92]或 JTAG[93]。一般并行方式总是在开发阶段才会使用。

这样分类的结果如图 5-90 所示。实际使用的接口方式不外乎方法 1 至方法 6。在以下各节中分别通过一个方框简图一一加以介绍。

例子中为简单起见，仅考虑微控制器板上是否有 CAL-RAM 这一特征。这样的简化忽略 CAL-RAM 存放在微控制器的内部还是外部 RAM 中，也不再进一步区分在哪种设备上实现（ECU 或微控制器）。

5.6.7.1　有内部 CAL-RAM 的串行接口（方法 1）

该串行接口使用 ECU 所带的串口（优点在于与车载应用相比，几乎不需要修改）（图 5-91）。CAL-RAM 在产品 ECU 中是不需要的。出于费用考虑，开发阶段使用的 ECU 可以选择带有额外存放 CAL-RAM 的微控制器型号，而在生产时选择相对便宜的型号。当然，如果不是这样的话，产品 ECU 的费用会升高。

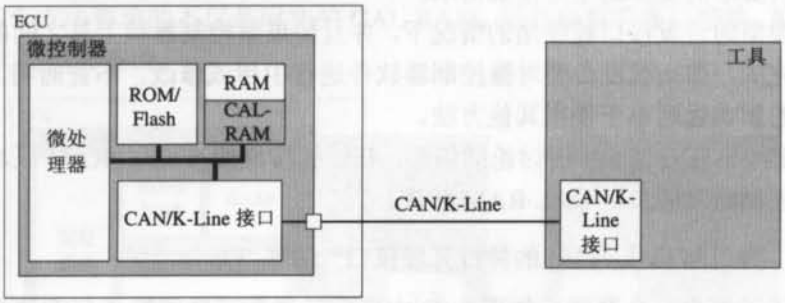


图 5-91　有内部 CAL-RAM 的串行接口（方法 1）

为节省费用，通常也要限制通信速率，导致在开发阶段得不到所需的高速传输功能。

当确实需要高速通信时，可以采用 K 总线[5]或 CAN 总线。

如果系统正常工作时也使用该接口进行的通信，如 CAN 接口，此时通信负载或许已经相当重。则可以采用增加第二个 CAN 接口的方法专用于开发工具的通信。

这两种情况下位的微控制器都要负责提供与上位工具软件通信所需的处理能力和内存。

通常在线标定受限于 CAL-RAM 区域的大小。这样做极大地限制了每次可以标定的特征值、曲线和图表的数目。CAL-RAM 容量的动态管理和分配可以部分解决这一问题，但是该方法也会消耗额外的微控制器资源。

CAL-RAM 管理方法在第 5.6.8 部分讨论。

5.6.7.2 有内部 CAL-RAM 的串行开发接口（方法 2）

相比产品 ECU，该方法中与内部 CAL-RAM 连接的串行开发接口的作用，是使 ECU 方面仅需要较小硬件变化。然而，开发接口的一些部件，如 NEXUS[92]或 JTAG[93]，被用于其他应用领域（如调试操作）。其他的微控制器接口都是针对具体的微控制器而设计的。在大多数情况下，在考虑到苛刻条件下的车内测试时，这些并不能实现所有需求。相应地，专门为车内应用配置的接口的变换必须在 ECU 自身实现。根据这一原则，一个解决方法是使用一种所谓的串行 ETK [94]，如图 5-92 所示。

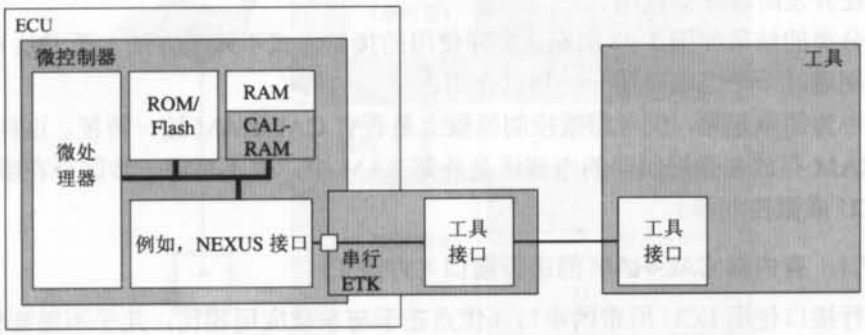


图 5-92 有内部 CAL-RAM 的串行开发接口（方法 2）

几乎没有例外，开发接口的传输率显著地高于产品接口的，因而实现了开发阶段在测量技术方面所要求的更多的全方位的需求。

在这种类型的开发接口被使用的情况下，并且如果微控制器和工具之间的通信利用一个硬件设备完成，那么就没必要对微控制器软件进行引申或修改。不言而明，这个方法对于运行时间的影响远远小于使用其他方法。

然而，正如早在方法 1 中所讨论的情形，在线校准继续被 CAL-RAM 区域的大小所限制，并且微控制器继续负担 CAL-RAM 管理。

5.6.7.3 有内部 CAL-RAM 的并行开发接口（方法 3）

该方法可以作为一个选择，如图 5-93 中所示。尽管其性能特点与方法 2 的相似，然而在 ECU 上必需的硬件修改比它们通过串口存取时更广泛。为此，方法 3 几乎没有实用价值。

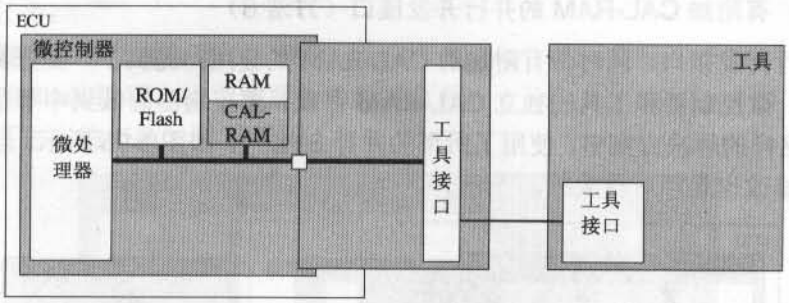


图 5-93　有内部 CAL-RAM 的并行开发接口 （方法 3）

5.6.7.4　有附加 CAL-RAM 的串行接口（方法 4）

在线校准的内部 CAL-RAM 的尺寸限制，可以通过在开发 ECU 中安装附加的 CAL-RAM 的容量来补偿，并且也通过使用以附加的 CAL-RAM 为特色的微控制器的开发样本来补偿，如图 5-94 所示。

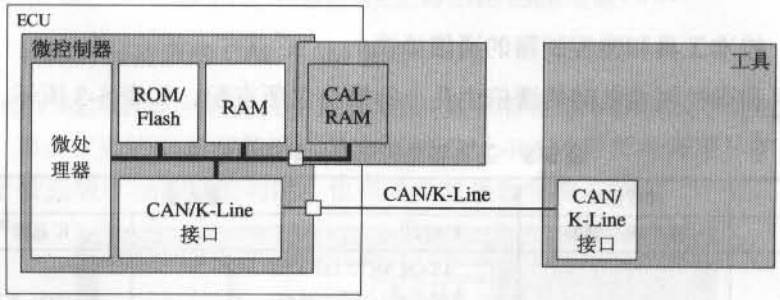


图 5-94　有附加 CAL-RAM 的串行试制接口（方法 4）

使用如图 5-94 所示的接口，同样存在着对于传输性能和微控制器负荷的限制。

5.6.7.5　有附加 CAL-RAM 的串行开发接口（方法 5）

关于传输率和微控制器负荷的限制，能通过该方法中所描述的串行开发接口的使用而被避免。图 5-95 中所示就是这样的解决方法，其中使用了串行 ETK[94]和附加的 CAL-RAM。然而，因为这个方法要求的不仅是附加的 CAL-RAM，而且也要求接口变换，所以要对 ECU 硬件进行较大范围的修改。

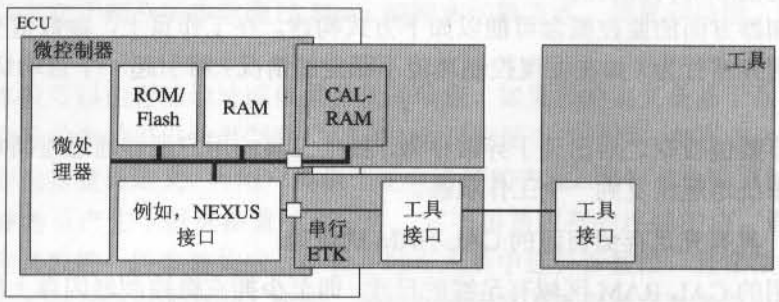


图 5-95　有附加 CAL-RAM 的串行开发接口（方法 5）

5.6.7.6 有附加 CAL-RAM 的并行开发接口（方法 6）

应用并行开发接口，同时带有附加的 CAL-RAM 的使用，提供了一些好处，如在线存取校准数据、微控制器和工具的独立 CAL-RAM 存取、微控制器的低到中等负荷和高的传输性能等。这样的解决方案中，使用了所谓的并行 ETK[95]，如图 5-96 所示。这里，对 ECU 硬件的必须修改也是相对较大的。

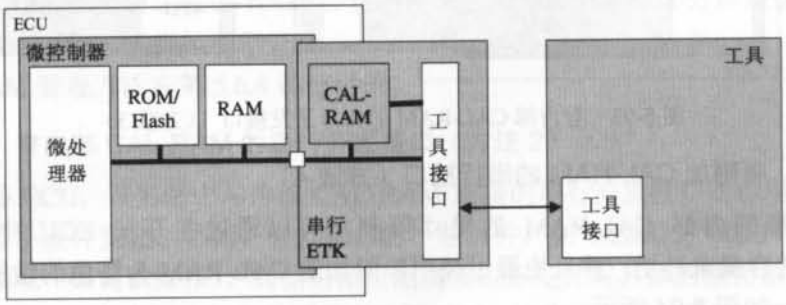


图 5-96 有附加 CAL-RAM 的并行开发接口（方法 6）

5.6.7.7 校准工具和微控制器的通信协议

校准工具和微控制器之间的通信由几个标准协议所支配，如表 5-3 所示。

表 5-3 工具和微控制器间的通信标准

接　口	并行接口	串行接口	
物理层	基于微控制器	CAN[2]	K 总线[5]
协议		ASAM MCD 1a：CCP/XCP[18] ISO：CAN[25]的诊断	ISO：KWP 2000[24]

5.6.8 CAL-RAM 管理

不考虑所选择的方法，微控制器一旦被初始化，就根据参考页参量开始运转。这样，没有被连接的校准工具，运行也是可能进行的。在初始化中，工作页被新的数据覆盖，这些新数据是从参考页中获得的。之后，即将被执行的程序可以使用从参考页或工作页中所获得的参量。这就要求行驶程序在工作页和参考页之间有一个切换，并且反之亦然。这能够通过修改微控制器的软件或硬件来实现，有些微控制器提供这一功能的完善的硬件支持。切换是由校准工具控制的。

在微控制器方面的监控概念可能以如下方式构建。在工作页上，参数值变化后行驶程序功能的任何异常行为（如在发现控制功能不稳定的情况）将引起一个自动切换到“紧急”运行状态。

如果在参数值改变之后出现了异常行为，校准工具的用户能够通过重新切换到系统参考页，使得系统迅速恢复到一个工作状态。

5.6.8.1 具有充足存储资源的 CAL-RAM 的管理

假如可用的 CAL-RAM 区域有足够的尺寸，即至少拥有微控制器闪存上的数据版本所必需的存储容量的尺寸，那么消耗在微控制器方面供 CAL-RAM 管理的空间，在本质上被

限制到三项任务：

（1）将整个参考页复制到工作页；

（2）通过 Flash 编程，将工作页备份到参考页；

（3）两页之间的切换（图 5-97）。

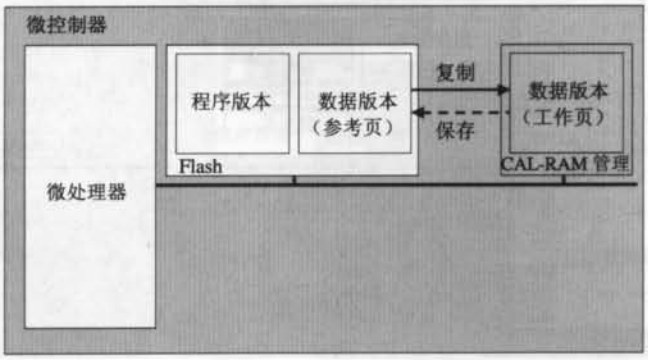


图 5-97　存储资源充足的 CAL-RAM 管理

5.6.8.2　有限存储资源的 CAL-RAM 管理

然而，如果 CAL-RAM 区域的大小是有限的，意味着将整个数据版本复制到 CAL-RAM 是不可能的，那么在微控制器和校准工具方面的 CAL-RAM 管理都需要被扩展——通过附加一个有利于数据版本段管理的功能，也就是参数集合（图 5-98）。

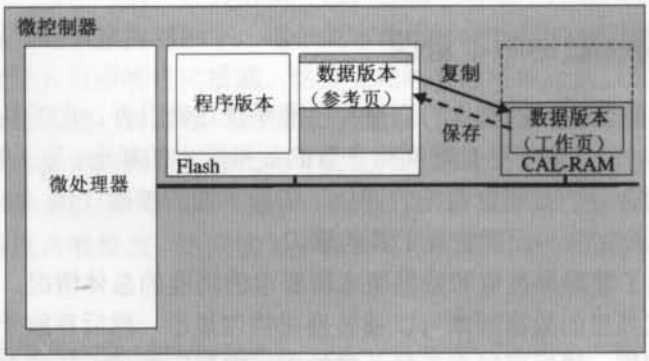


图 5-98　存储资源有限的 CAL-RAM 管理

这样，只有对于部分的参数通过 Flash 编程和切换方式，使复制功能、在线校准才成为可能。

数据版本段可以由存储地址或依据参量来确定。如果段的定义是基于存储的，那么连续一整片闪存段的参数可以被复制到 CAL-RAM 以供随后的校准，如图 5-98 所示。

数据段面向参量而定义，为用户提供了更多方便的方法，因为它允许更加灵活地编程。原因之一是参考页产生于所选参量的存储区，它不再是闪存的连续的块，而是如图 5-99 所示由闪存中选取的一段参数构成。可以在校准工具中选择参数段，且在微控制器中，通过指针表的方式进行随后的复制。这样，微控制器程序不再直接存取参考页上的参数（在

图 5-99 中被标记为①），而是在指针表标签的辅助下间接存取（在图 5-99 中被标记为②）。在这样描述的过程中，校准工具可以在参考页存取参数（通过③）和存取工作页（通过④）之间切换。

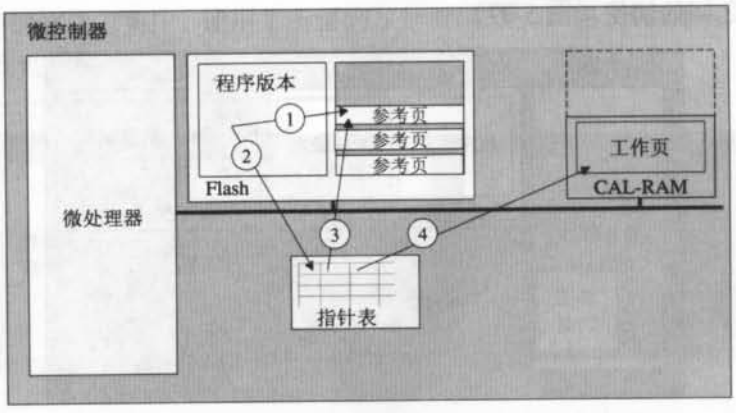


图 5-99 管理参数段的指针表[86]

如果指针表自身被存储在微控制器的闪存中，那么对于参数段设置的每一次修改，都将要求相应的 Flash 段被重新编程，并且参数段设置将在行驶程序执行的过程中保持不变。

一个可选择的解决方案是在微控制器的 CAL-RAM 区域中存放指针表。这样，当行驶程序改变时，指针表也可以动态地改变。

5.6.9 参量和数据版本管理

到目前为止，除讨论过的基本的测量和校准系统功能以外，在工具方面还必须支持进一步的功能。这些功能的例子是处理面向参数的数据版本的导出、导入和数据版本的合并、程序和数据版本的合并、被测量数据的评估、依赖参数的校准（图 5-57）、虚拟信号（图 5-58）的计算，以及文档和配置管理工具的接口。

图 5-100 提供了管理参数值和数据版本所要求的功能的总体情况。在来自描述文档的信息的帮助下，工具里的数据版本可以被转换成物理显示，然后存放于数据库中。这些物理参数值能够被编辑，然后可以在各种各样的应用中用于更进一步的处理。使用软件组件数据模型的说明书，有利于软件组件的数据模型的参数化；以软件组件数据模型的设计为基础，使得在源代码中软件组件的参数化成为可能，相反的做法也是可能的。

5.6.9.1 二进制程序版本文件和数据版本文件的参数化

包含产生最终的数据版本和二进制文件的参数化。程序版本和数据版本都以生产和服务应用中所要求的二进制格式提供。

这种方法的好处是校准系统能够提供整体所必需的功能，它属于系统的标准特性集合。跨越公司的边界，校准任务也有可能在多个企业间分布进行。这样，编程和数据版本的交换就是单向的，即仅限于从供应商到汽车制造商的单向托付。

然而，二进制文件格式缺点在于限制了物理水平的数据记录管理，以及项目间的简单

的利用。另一个缺点可以在事实中看到，一旦校准规程完成了，关于程序版本的优化就不能再被执行。对校准规程需要的可能的软件结构的修改，这些至少部分在产品版本中已经具有。

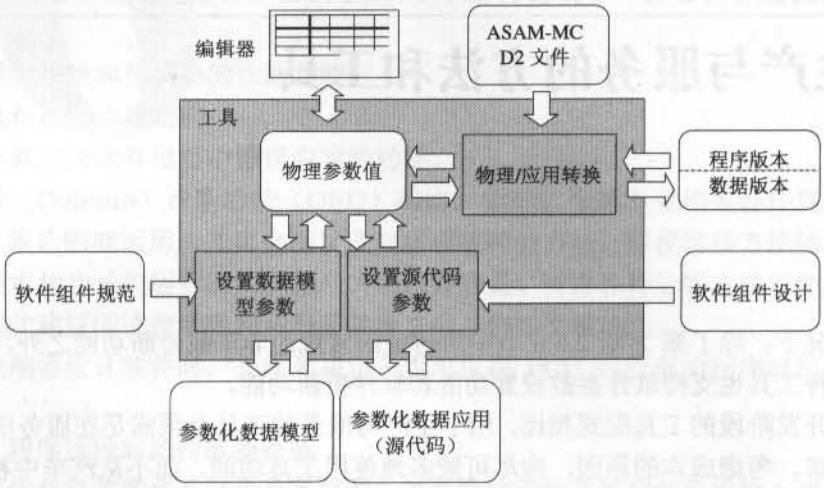


图 5-100　参数值和数据版本的管理

5.6.9.2　模型或源代码的参数化

上述缺点可以通过使用另一种方法来补偿。在这一方法中，校准系统导出数据版本，并且在关于数据模型的信息的辅助下，这个版本被用于源代码或模型层的校准。

随后，可以执行大范围的优化措施。这些变化的范围可能会从以运行时间或存储容量优化为目标的最优化存储，到 X 轴点的数量的改变，到特征值、曲线和图的量化、值域或存储段的优化。经过优化的模型或源代码可能选择使用合适的编译器被重译，且随后进行必要的质量检查，形成供生产和服务使用的版本。因为，在这之后，参数值将不再修改，或仅仅在一定的程度内被修改，这些优化步骤仅仅是在用于生产和服务的数据版本的发布前才是可能的。

5.6.10　实验的设计和自动化

通过自动化的接口，如 ASAM MCD3[17]，一个高级自动化系统控制多种功能。这些便利不仅仅是离线任务的自动化（如记录的测量数据的评估）而且也是在线进程的自动化（如在实验室车辆或试验台中长时间运行的调整程序），正如之前在 5.5.4 节所讨论的[29, 30, 31]。

许多相关活动，如被测数据分析的自动化或优化参数值的确定，是当前研究与开发工作的主要内容。

尽管这些和相关课题的意义巨大，但是很抱歉，因为详尽地讨论这个课题已经远超出了本书的范围。

第 6 章 Chapter 6

用于生产与服务的方法和工具

许多情况下，除了第 2 章 2.6.6 节中所讨论的常规的非车载诊断功能之外，用于生产与服务的多种工具也支持软件参数设置功能和软件更新功能。

然而与开发阶段的工具配置相比，用于生产与服务的工具必须满足在服务中的一些额外的特殊要求。考虑成本的原因，应尽可能多地使用工具功能，而不从汽车中移除 ECU。通过第 2 章图 2-67 所示的汽车非车载诊断界面，服务工具与装载在汽车上的 ECU 产品之间经常需要有标准通信路由。另一个重要的方面涉及了在服务维修站所执行的软件更新。在这种情况下，为了不需要从物理上移除 ECU，必须保证软件工具的高度可用性，以响应不正常的运作或功能性的异常中断。

工具与 ECU 之间的标准通信协议（如关键词协议 2000[25]或 CAN 诊断协议[26]），以及相应工具（如 ASAM-MCD[17]）的标准化数据库正越来越多地取代一些专用的方法。图 6-1 显示了基于 ASAM-MCD[17]的非车载诊断系统的结构。ASAM-MCD[17]标准追求的是定义一种统一的标准软件结构，该标准软件结构包括了汽车软件开发、生产和服务中所配置的一些测量、校准和诊断工具。

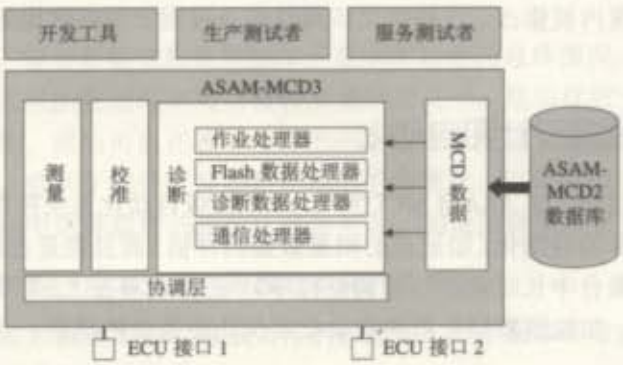


图 6-1　基于 ASAM-MCD 的诊断系统结构

6.1　非车载诊断

除了一些基本的功能之外，前面 2.6.6 节中所讨论的非车载诊断系统还提供了以下功能：

（1）设定系统和传感器的诊断功能；

（2）执行器的诊断功能；

（3）下载并清除存储器中错误内容的功能。

与车载（Onboard）诊断功能（OBD）相比，非车载诊断不受相关硬件资源的约束。因此，非车载诊断能运用一些可全面发现并修理故障的方法，即使这些方法随之产生的首要条件是要求相当长的运行时间和相当大的内存容量。讨论各种诊断方法可能超出了本书的范围，因此建议阅读参考资料和最新参考文献，例如文献[52]。

为微控制器设计软件时，必须要决定是否可以在 ECU 运行驱动程序时执行非车载诊断功能。

举例：程序运行时的传感器诊断

传感器诊断经常需要获取传感器信号。传感器信号就是微控制器的输入变量，以及在线传输到非车载诊断系统的测量值。当正常的程序被执行时，诊断检测器能激活微控制器中所装载的功能。

诊断检测器经常用于接受其他诊断测量设备的连接请求。诊断测量设备即所谓的诊断检测仪表，允许同步记录其他的外部信号。因此，在与图 5-84 所示不同的测量方法的帮助下，可以完成传感器诊断。

举例：程序停止运行时的执行器诊断

相比较而言，执行器的诊断经常支持非车载诊断系统，能直接给出执行器设定值的定义，这定义就是期望的微控制器的输出值的规格说明和激励。

最后，在正常操作期间计算设定值的那些程序功能至少要被中止。程序功能中止通过转换到用于执行器诊断的特别操作状态就可以实现，这种转换是由诊断检测器引发的（见图 4-23）。

和校准情况一样，完成实现方式到物理显示的转换，要在测量时或预定义的内部微控制器信号的出现时就装载诊断工具，反之亦然。

6.2　软件功能参数设置

对于当今汽车制造来讲，用于汽车软件生产和服务的总体情形可以通过大量的汽车型号表现出来，产生这种现象的原因同样很多。首先，任何汽车用户能通过选择多个单一装备选项来装配他所定制的汽车。其次，不同国家汽车市场中本土的多种规章和要求或多或少导致了一些具体国家的汽车要求。第三个影响因素包括允许相同汽车的一些驾驶者在随车携带的内存中存储他们自己的一些设置以供即时调用，例如后视镜的位置、喜爱的电台配置、空调的设置。这些具体的驾驶者设置也被称为个性设置[96]。

用户的汽车装备、具体国家的汽车装备以及个性设置，这些都导致了不同的汽车及其功能型号。因此，多种软件型号就变得必不可少。

在汽车生产和服务中，专用的软件配置参数可以控制前面所提及的一些型号。因此，这可能会大大削减汽车电子系统所处理的硬件型号的数量。

所有这些逻辑上都需要设计到处可配置的硬件。几乎在汽车生产线的末端，例如在汽车安装 ECU 之前极短的时间里，通过一个合适的参数设置程序就可选择一个特定软件型号。另一个方法是延迟这种选择直到它可以在服务维修站执行，例如在汽车里已经安装的一个 ECU 上。

对于个性化操作配置，必须设置、存储、选择、并且激活这些具体用户的软件参数。这可在服务维修站利用一个服务测试功能来完成，或由用户自己通过使用用户界面所提供的功能来完成。

为了防止出现错误的规格说明或非法的参数设置，在软件体系结构设计之初就必须考虑这个软件参数设置程序。

如图 6-2 所示，在汽车的逻辑体系结构上观察这项任务，比在技术体系结构上有明显的优势。

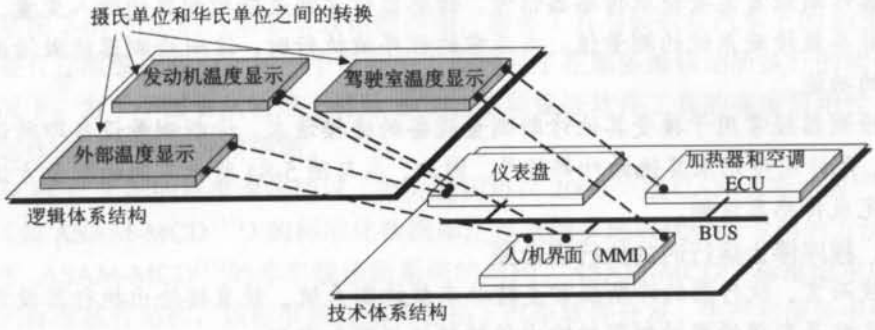


图 6-2　负责温度显示的 ECU 及其功能[96]

举例：摄氏度数和华氏度数之间的温度显示单位转换

作为温度显示测量单位的参数选项可以看作是一个相关事件。其目的是想通过单个软件参数，在摄氏度数和华氏度数之间可转换所有温度显示的功能。举例来说，这个软件参数就影响了发动机、车壳内外温度的显示功能。如图 6-2 所示，在 ECU 这边，转换这些显示参数对仪表盘、加热器和空调 ECU 以及人机界面（MMI）都有影响。由于温度可在各种显示单元进行显示，例如在仪器组和 MMI 里显示发动机的温度，每个独立的 ECU 参数选择都会考虑到所设定的不同值。仅仅通过在功能级设置参数就能防止所引起的不精确性。

正如 5.4.4 节所讨论的，和技术实现有关的几个选择是有区别的。软件参数值能够存储在如图 5-69 所示的 Flash 存储器中，或者被写入如图 5-70 所示的微控制器的 EEPROM 中。在这两种情况下，可以使用生产和服务工具的功能来改变软件参数值。

和校准（见 5.6 节）想调整整个数据版本相比，基于服务的软件功能参数只调整那些所选择的且存储在可编程内存里的软件参数，例如产品 ECU 上的 EEPROM 或 Flash 内存里所存储的软件参数。

在标准实行过程中，也就是在驱动程序的中断期间，可充分完成这些离线参数的选择。通过转换到用于软件参数设置的特殊操作状态，由参数设置工具的触发就可以完成这些离线参数的选择（见图 4-23）。已经证明，参数设置程序类似于所选择的参数在离线校准里所使用的程序。

6.3　通过 Flash 编程的软件更新

作为程序版本和数据版本的存储方法之一，Flash 存储器也在相关产品 ECU 中越来越多地被使用。这就为 ECU 软件更新开启了大门，即尽可能地通过汽车的非车载诊断界面，通过重新编制 Flash 存储器就可以完成 ECU 软件更新。这样不需从汽车中移除 ECU，就可以方便地实现软件更新。不用替换物理的 ECU，显然可以节约成本。

本节随后的内容包括一些 Flash 编程技术的首要前提，讨论一个基于非车载诊断界面的 Flash 编程程序。为了简单明了地论述，这部分讨论省略了一些可选择的用于提高 Flash 编程速度的扩展描写。

6.3.1　Flash 存储器的擦除和编程

2.3 节和 5.6 节中提到，现今 Flash 技术只用于所有 Flash 存储器片断（Segment）的擦除和重新编程。一个片断或 Flash 片断指的就是物理上连续的、可作为一个实体被擦除和编程的最小的 Flash 存储单元。但 Flash 编程在 Flash 内存片断内的擦除步骤和编程步骤之间是有差异的。

同样也要注意到，技术上的一些障碍阻止了 Flash 片断内一个程序的同步执行，而同一 Flash 单元的其他 Flash 片断却可以被重新编程。鉴于这个原因，在实际的 Flash 编程期间，对于特定的 Flash 单元，需要控制程序流的程序必须要被擦除或移动到其他存储位置，例如移到另一个 Flash 单元或微控制器上装载的空的 RAM 片断。

排除了这些相关的前提条件，以下的章节将会忽略有关未来微控制器或具体存储器区别上的一些细节。

6.3.2　诊断界面的 Flash 编程

汽车生产和服务经常要求减少 Flash 编程次数是有理由的。因为与较大的 Flash 内存片断相联系，受限的传输速度和/或非车载诊断界面的性能联合起来就导致了相当长的 Flash 编程时间。通过减少要求重新编程的 Flash 片断数，也就是利用有功能导向的 Flash 编程，或通过程序版本和数据版本的分开编程，是有可能减少 Flash 编程次数的。在实际的操作过程中，程序版本早在 ECU 生产期间就经常被编好程序了，而数据版本（汽车和型号说明）在新汽车非常精确地到达生产线的末端时就被编程了。最后，软件开发必须存储各种软件功能，以及不同 Flash 片断中的程序版本和数据版本。

与 Flash 编程程序和 Flash 装载程序（见图 1-22）一起，所有负责微控制器和 Flash 编程工具之间通信的微控制器部分必须存储在 ROM 或第三 Flash 片断中。存储在 ROM 中的

基本程序叫做启动模块，与之相对应的存储在 Flash 存储器中的叫做引导模块。图 6-3 描述了完整的程序，它可分为四个部分：启动模块，引导模块，程序版本和数据版本。启动模块和引导模块的结合使得微控制器的软件功能变为可用，这个软件功能是车外诊断界面的 Flash 编程所要求的。

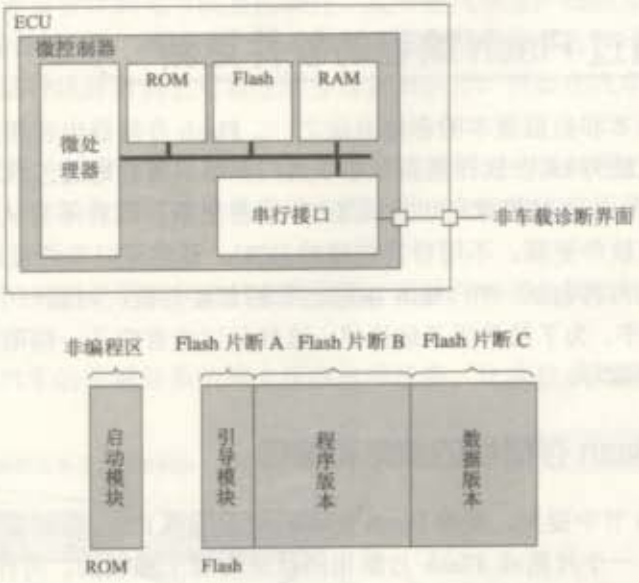


图 6-3 启动模块、引导模块、程序版本和数据版本的内存分配

启动模块和引导模块的分离是有很多好处的。首先，存储在 Flash 存储器中的引导模块本身可以被单独重新编程。引导模块切换程序和随后的 Flash 编程会在 6.3.5 节中详细讨论。其次，对特定的 Flash 编程的当前状态进行非易失性保存，引导模块是理想的存储地。这意味着在编程中断或异常事件中能方便地进行重新启动。相比起来，启动模块中不能变更的基本功能程序可以存储在更节省成本的 ROM 里。

然而，作为程序版本和数据版本的一部分，驱动程序被存储在不同的存储器片断内。下面的内容将在四个程序部分（启动模块、引导模块、程序版本和数据版本）中进行区分。

6.3.3 安全性要求

Flash 编程工具引发了微控制器向软件更新操作状态的转变。

5.6 节所讨论的真实性检查，例如和发动机 ECU 相联的发动机停顿检查，必须在驱动程序结束之前完成，并随后转换到软件更新操作状态。然而，汽车软件生产和服务需要额外的安全检查。产品责任考虑要求尽可能地防止任一未被授权的，也就是非法的 Flash 编程，或转让、传输和/或安装操作的程序版本或数据版本的 Flash 编程。至少可以发现和检验出上述的传输和/或安装的 Flash 编程。

鉴于这些原因，Flash 编程访问通常受到两个不同的加密方案的保护，这两个加密方案称为身份认证和签名认证。图 6-4 显示了 Flash 编程工具和微控制器之间的通信程序流。

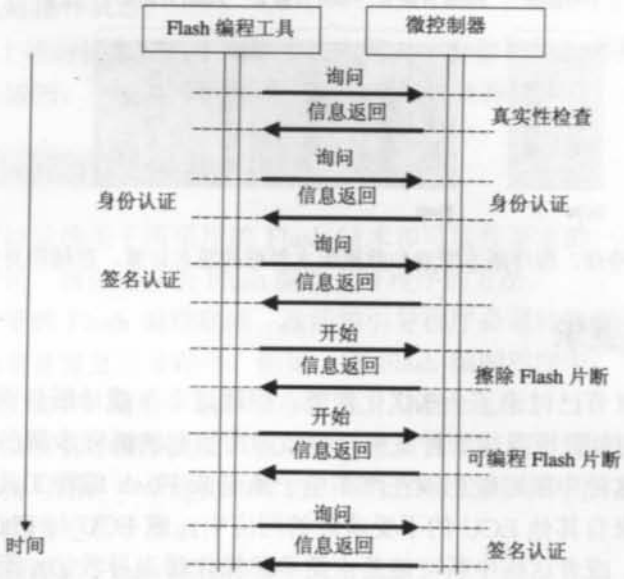


图 6-4　保护 Flash 编程防止未授权的访问的安全措施

1. 身份认证

真实性检查之后，要执行访问权利的实际认证，这是身份认证检查。该程序使用第一个数字密码来确认 Flash 编程工具的各个用户是否有使用软件更新的权利。

2. 编制的新程序版本或数据版本的签名认证

检查所编制的新数据版本或新程序版本的数据一致性，即签名认证。该程序使用第二个数字密码来确认各自的程序版本和/或数据版本是否与 ECU 硬件相匹配。认证也检查所编制的程序版本和/或数据版本，自从汽车制造商把它发布到生产车间或者服务商店以来，是否被未授权修改了。

3. Flash 片断的擦除和编程

引导模块允许在有疑问的 Flash 片断之上，且仅在两个认证步骤（身份认证和签名认证）已经成功结束了之后，继续进行实际的擦除和编程。

4. 新编制的程序版本和数据版本的签名认证

随着 Flash 编程期的结束，微控制器使用刚编制的实际程序版本和数据版本来计算签名，并检查可能的编程错误。一旦签名认证结束，签名计算的实际结果会存储在 Flash 存储器中。这可通过存储特殊的内存结构来完成，该内存结构在 Flash 存储器中是程序版本和数据版本的一部分（图 6-5）。这些就是所谓的程序版本和数据版本的管理。成功的签名认证是引导模块激活新驱动程序的信号。

可用的加密方案超过了本书的讨论范围，建议阅读参考文献[97]。

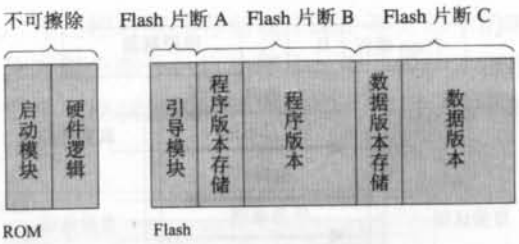


图 6-5 硬件管理、程序版本管理和数据版本管理的签名计算、存储和身份认证

6.3.4 可用性要求

尽管前面的一些章节已讨论了一些优化措施，但通过非车载诊断界面的 Flash 编程可能会持续相对较长的时间，而且由一种或另一种故障所引起的编程步骤的一些中断在任何时间都可被预测到。这种中断可能出现在汽车电子系统或 Flash 编程工具的电力故障的假象中，出现在网络上来自其他 ECU 的不受欢迎的回应中，或 ECU 与 Flash 编程工具之间的通信线路的中断中。或者这些中断可能是由简单的操作错误导致的，而失败的身份认证和签名认证也会导致 Flash 编程期异常中断。

由于以上原因，在任何情形下，Flash 编程的可用性在 Flash 编制程序的设计过程中占有优先地位。这个要求是可以被满足的，例如在一个程序异常中断之后只要重新启动该程序序列即可。下面的例子就描述了这样一个程序序列。

举例：程序版本和数据版本的 Flash 编程过程

图 6-6 显示了图 6-5 所示的程序版本和数据版本的一个典型的 Flash 编程过程。

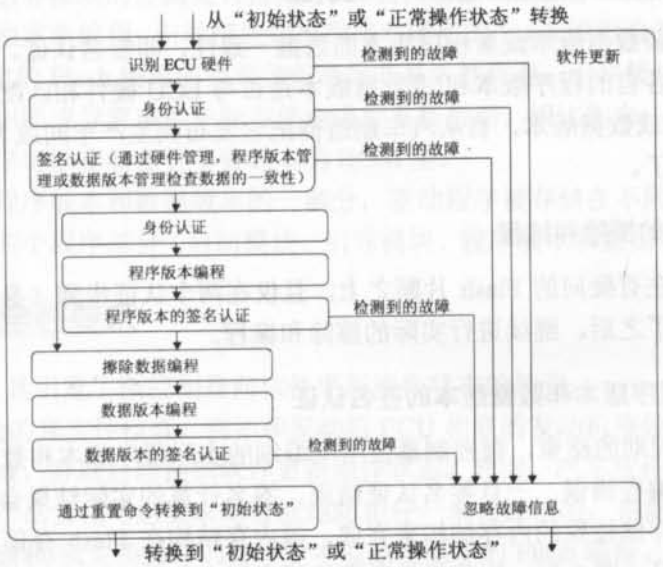


图 6-6 Flash 编制程序和数据版本时的引导模块的状态和转换

在 Flash 编程成功结束时，该编程工具发布了一个重置命令，该命令使得微控制器转

换到了一个一般的操作状态。

迄今为止，上述讨论都忽略了引导模块向另一个存储模块的转换，该转换在实际 Flash 编制程序中是必需的。下面一节既强调了编制引导模块程序本身，也强调了它的转换。

6.3.5 引导程序转换和 Flash 编程

考虑到先前讨论的关于所采用的 Flash 技术和可用性要求的一些首要条件，为了完成这部分内容的讨论，将详细分析 Flash 编制引导程序的方法。

首先，在期望的 Flash 编程期间，激活的引导程序必须转换到微处理器的另一个内存单元中。这意味着要重置引导程序。例如，在 Flash 编制程序中，可以将引导程序复制到未占用的 RAM 区域中。随后，引导程序可在 RAM 内运行。

即使 Flash 编制引导程序失败，也必须使重新启动编程步骤成为可能。在编程异常中断后，为了仍保持可用，一个避免故障的引导程序将发挥其作用。把复原和重建命令应用于引导程序就可用满足这个要求。

举例：启动程序的 Flash 编程步骤

使用如图 6-7 所示的程序步骤可以满足上述一些要求，但旧的启动程序和新的启动程序之间是有区别的。

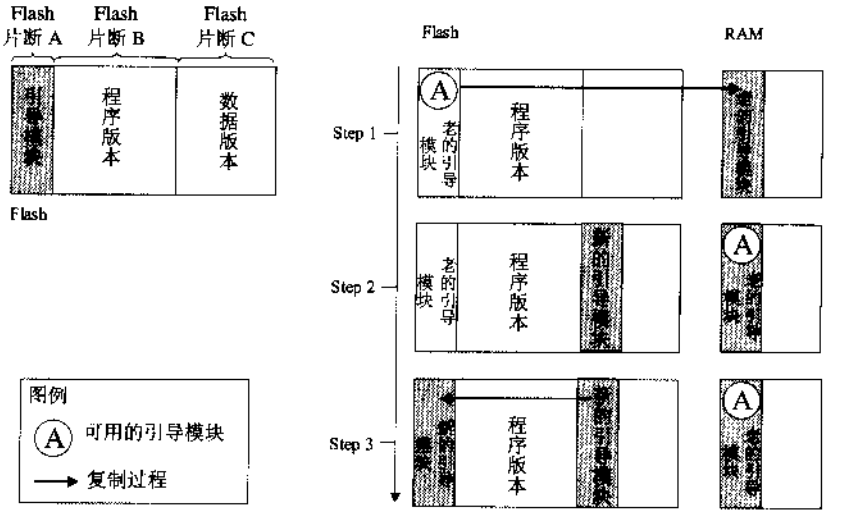


图 6-7　引导程序 Flash 编程步骤

程序主要分为三步：

步骤 1：将旧的引导程序复制到 RAM 的空白区。

步骤 2.1：使旧的引导程序在 RAM 中可用，在 Flash 内存中不可用。

步骤 2.2：将新的引导程序临时存储在 Flash 片断 C 中。这一步包含了“擦除 Flash 片断 C”程序，“在 Flash 片断 C 中编制新的引导程序”以及“Flash 片断 C 中检验新的引导程序签名”。这些操作中的任一操作发生程序异常中断时，Flash 片断 A 中的有效的旧引导

程序能用于重新启动 Flash 编程。

步骤 3：通过将 Flash 片断 C 复制到 Flash 片断 A 来编制新的引导程序。这一步包含了“擦除 Flash 片断 A”程序，“通过将 Flash 片断 C 复制到片断 A 来编制新的引导程序”以及“在 Flash 片断 A 中检验新的引导程序签名”。这些操作中的任一操作发生程序异常中断事件时，Flash 片断 C 中的有效的旧引导程序能用于重新启动 Flash 编程。

Flash 内存中需要分别标识有效的启动程序。这种有效性标识本身必须存储在 Flash 内存中，且不会消失，从而使这个信息成为可能重启的基础。

重新启动程序紧随在 Flash 片断 A 中激活新引导程序，以及在 RAM 中锁死引导程序之后。最后一步包含了如图 6-6 中所示的 Flash 编制数据版本程序。

6.4 汽车电子系统的启动和测试

在生产过程的最后阶段，新的汽车第一次被送去工作车间。在工作车间第一次启动以及测试完全集成的汽车电子系统，以及测试每一个非车载诊断界面。类似的情况是，在汽车的维修服务站中，单个汽车电子的组成部分偶尔也可以在维修服务站进行替换和测试。

所有应用的方法和工具都是基于 6.1 至 6.3 节中所讨论的功能。而一些相关的应用案例则是对于特定汽车所替换的电子组成部分的配置，或是对于 ECU 中程序的 Flash 编程。

第 7 章 *Chapter*

总结和展望

电子系统和软件已经成为现代汽车中许多功能的不可缺少的组成部分。我们有足够的理由相信，将会有越来越多的汽车功能通过软件来实现。

通过电子系统和软件来实现汽车功能需要应用多个不同学科领域的基本知识。在本书的第 1 和第 2 章中，对于汽车电子系统的总体概况，控制系统、实时系统、分布式网络系统、系统的可靠性以及与安全性相关的系统的基本知识做了重点的介绍。

电子系统和软件的开发必须被看作是整个系统开发过程中的工程学科分支。对于以硬件为主要特点的汽车功能、系统或者组件的理解，也必须要考虑到软件作用。汽车功能的开发通常是在多个不同的部门和企业间分布完成，这也导致汽车软件的开发也是由多个不同的开发团队参与共同完成，每一个团队负责其中的一部分（软件组件），例如实时操作系统。

因此，对于系统和软件的开发过程需要一个一致的开发过程模型。开发过程模型应该考虑到开发过程的每一个步骤，包括从用户的需求分析、系统功能的描述、系统的设计、系统的集成、系统的标定，直至系统的验收测试。进一步地，还需要考虑到汽车的整个生命周期，例如在汽车服务阶段所需的监视和诊断功能也必须在开发过程中尽早地考虑到。

OSEK、ASAM 和 ISO 标准为定义与硬件无关的应用软件和平台软件之间的接口提供了基础。因此，现在的系统越来越多地不像以前那样，是把硬件与软件以固定的方式绑定在一起的。

在本书第 3 章和第 4 章中讨论的过程，用于支持汽车电子系统和软件的开发和长期的维护工作。

基于模型的汽车电子系统功能、软件功能以及环境的描述，有助于简化理解，并且还为开发过程中各个阶段提供了许多的优点。

汽车功能模型可以用于评价汽车功能的各种不同的技术实现，也可以用于分析和功能描述阶段，并为仿真和快速原型的方法提供了基础。此外，在合适的代码生成工具的帮助下，功能模型被用于直接生成汽车电子控制单元的软件组件。

在集成和测试阶段，实验汽车和测试台中也需要应用功能模型和环境模型。通过应用这些模型，能够考虑到对于多个企业间系统集成和测试任务的一些特殊要求。

在生产和服务阶段，功能模型有助于建立用于诊断、参数化以及软件升级更新的有效的技术和方法。

通过仿真方法、开发过程的自动化以及把某些开发的步骤从真实的汽车中移到测试台和实验室中，使得在降低开发风险的同时，有可能优化质量和成本。与在道路上真实汽车中的实验相比较，我们在测试方面可以具有更大的自由度和更高的效率。

然而，仿真方法只能回答我们已经提出的问题。与此相反，在道路上真实的汽车中的测试能够回答事先并没有想到的问题。

因此，在道路上的真实汽车中的测试是不能缺少的，因为只有通过在真实汽车中进行的验收测试，汽车功能的有效性才能得到验证。在真实汽车中进行的测试对于测试工具也提出了很高的要求。例如，支持与控制器之间的非车载接口，应用于恶劣环境条件中的移动测量技术，适合于汽车环境的便于用户操作的界面和可视化显示。

本书的第 5 章和第 6 章阐述支持核心开发过程，以及支持生产和服务过程的方法与工具。

涉及多个子系统的汽车功能的开发越来越多。例如，通过汽车与周围环境之间的无线传输系统的联网，能够实现许多新的汽车功能，这些功能包括从汽车相互碰撞前的识别直至预测诊断功能。因此，网络化的概念必须包含更为广泛的内容。

除了汽车系统，汽车周围的环境系统也必须被考虑进来。因此对于分布式和网络化系统的分析方法，系统功能的描述方法以及系统的集成方法提出了更高的要求。

正是由于这些大量的要求和相互间的影响，汽车功能的开发工作是一项很难的任务。必须通过应用尽可能少和尽可能简单的开发模式，来降低开发工作的复杂度。这种开发方式必须通过一致的开发过程、方法、工具和标准，来支持汽车的整个生命周期。而这些开发过程、方法、工具和标准的进一步发展需要汽车制造商、汽车零部件的供应商和开发工具制造商之间的进一步紧密合作。

因此，汽车软件工程学科的发展在未来将提供更多的发展机会和挑战。

参考文献

[1] Robert Bosch GmbH (Ed.), *Konventionelle und Elektronische Bremssysteme*, Robert Bosch GmbH, Stuttgart, Germany, 2002.

[2] ISO International Organization for Standardization, ISO 11898: Austausch digitaler Informationen; Controller Area Network (CAN) für schnellen Datenaustausch, 1994.

[3] Lapp, A., Torre Flores, P., Schirmer, J., Kraft, D., Hermsen, W., Bertram, T., and Petersen, J., "Software-Entwicklung für Steuergeräte im Systemverbund—Von der CARTRONIC-Domänenstruktur zum Steuergerätecode" ("Software Development for Networked Electronic Control Units—From the CARTRONIC Domain Structure to Code for Electronic Control Units"; original German), in Proceedings of the 10th International Congress "Electronic Systems for Vehicles," Baden-Baden, Germany, September 27–28, 2001, ed. by VDI Society Vehicle and Transportation Technology, VDI-Berichte 1646, VDI-Verlag, Düsseldorf, Germany, 2001, pp. 249–276.

[4] Robert Bosch GmbH (Ed.), *Kraftfahrtechnisches Taschenbuch*, 24th Edition, Vieweg-Verlag, Wiesbaden, Germany, 2002.

[5] International Organization for Standardization, ISO 9141: Straßenfahrzeuge; Diagnosesysteme; Anforderungen für den Austausch digitaler Informationen, 1992.

[6] Robert Bosch GmbH (Ed.), *Motormanagement ME-Motronic*, Robert Bosch GmbH, Stuttgart, Germany, 1999.

[7] Robert Bosch GmbH (Ed.), *Otto-Motormanagement: Grundlagen und Komponenten*, Robert Bosch GmbH, Stuttgart, Germany, 2002.

[8] Robert Bosch GmbH (Ed.), *Elektronische Dieselregelung EDC*, Robert Bosch GmbH, Stuttgart, Germany, 2001.

[9] Frischkorn, Hans-Georg, Negele, Herbert, and Meisenzahl, Johannes, "The Need for Systems Engineering. An Automotive Project Perspective," in Proceedings of the 2nd European Systems Engineering Conference (EuSEC), Munich, Germany, September 13–14, 2000 (keynote, on CD ROM), Herbert Uth Verlag, Munich, Germany, 2000, ISBN 3-89675-935-3.

[10] Fuchs, M., Lersch, F., and Pollehn, D., "Neues Rollenverständnis für die Entwicklung verteilter Systemverbunde in der Karosserie- und Sicherheitselektronik ("New Way to Handle the Development of Distributed Systems in Body and Safety Electronics"; original German), in Proceedings of the 10th International Congress "Electronic Systems for Vehicles," Baden-Baden, Germany, September 27–28, 2001, ed. by VDI Society Vehicle and Transportation Technology, VDI-Berichte 1646, VDI-Verlag, Düsseldorf, Germany, 2001, pp. 135–147.

[11] Eppinger, Andreas, Dieterle, Werner, and Bürger, Klaus Georg, "Mechatronik—Mit

ganzheitlichem Ansatz zu erhöhter Funktionalität und Kundennutzen," ATZ/MTZ special issue *Automotive Electronics*, September 2001, pp. 10–18.

[12] Stevens, Richard, Brook, Peter, Jackson, Ken, and Arnold, Stuart, *Systems Engineering. Coping with Complexity*, Prentice-Hall, Upper Saddle River, NJ, 1998.

[13] CMMI® Capability Maturity Model Integration®, http://www.sei.cmu.edu/cmmi.

[14] International Organization for Standardization/International Electrotechnical Commission, ISO/IEC 15504-1: Information Technology—Software Process Assessment—Concepts and Introductory Guide, 1998.

[15] V-Modell—Entwicklungsstandard für IT-Systeme des Bundes. Vorgehensmodell Kurzbeschreibung, 1997, http://www.v-modell.iabg.de/vm97.htm.

[16] OSEK Open Systems and the Corresponding Interfaces for Automotive Electronics, http://www.osek-vdx.org.

[17] ASAM Association for Standardisation of Automation- and Measuring Systems, http://www. asam.de.

[18] Deutsches Institut für Normung e.V., DIN 19250—Grundlegende Sicherheitsbetrachtungen für MSR-Schutzeinrichtungen, 1989.

[19] International Electrotechnical Commission, IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, 1998.

[20] Bundesgesetzblatt, "Verordnung über die Inkraftsetzung der ECE-Regelung Nr. 79 über einheitliche Bedingungen für die Genehmigung der Fahrzeuge hinsichtlich der Lenkanlage" (Verordnung zur ECE-Regelung Nr. 79), Part 2, 1995.

[21] DaimlerChrysler AG, "Übereinkommen über die Annahme einheitlicher technischer Vorschriften für Radfahrzeuge, Ausrüstungsgegenstände und Teile, die in Radfahrzeuge(n) eingebaut und/oder verwendet werden können, und die Bedingungen für die gegenseitige Anerkennung von Genehmigungen, die nach diesen Vorschriften erteilt wurden, ECE-Regelung Nr. 13, Einheitliche Vorschriften für die Genehmigung von Fahrzeugen der Klassen M, N und O hinsichtlich der Bremsen," DaimlerChrysler AG, Stuttgart, Germany, issue 2000-08-31.

[22] Eckrich, Michael, and Baumgartner, Werner, "By-Wire überlagert Mechanik," *Automobilentwicklung*, September 2001, pp. 24–25.

[23] IBM International Technical Support Organization, Redbook "Business Process Reengineering and Beyond," September 28, 2001, http://www.ibm.com/support.

[24] Bertram, Torsten, Opgen-Rhein, Peter, "Modellbildung und Simulation mechatro-nischer Systeme—Virtueller Fahrversuch als Schlüsseltechnologie der Zukunft," ATZ/MTZ special issue *Automotive Electronics*, September 2001, pp. 20–26.

[25] International Organization for Standardization, ISO 14230: Road Vehicles—Diagnostic Systems—Keyword Protocol 2000, 1999.

[26] International Organization for Standardization, ISO 15765: Road Vehicles—Diagnostic Systems—Diagnostics on CAN, 2000.

[27] Lange, K., Bortolazzi, J., Brangs, P., Marx, D., and Wagner, G., "Herstellerinitiative Software" ("Manufacturer Initiative Software"; original German), in Proceedings of the 10th International Congress "Electronic Systems for Vehicles," Baden-Baden, Germany, September 27-28, 2001, ed. by VDI Society Vehicle and Transportation Technology, VDI-Berichte 1646, VDI-Verlag, Düsseldorf, Germany, 2001, pp. 183–199.

[28] Erben, Meinhard, Fetzer, Joachim, and Schelling, Helmut, "Software-Komponenten—Ein neuer Trend in der Automobilelektronik," ATZ/MTZ special issue *Automotive Electronics*, September 2001, pp. 74–78.

[29] Gschweitl, Kurt, Pfluegl, Horst, Fortuna, Tiziana, and Leithgoeb, Rainer, "Steigerung der Effizienz in der modellbasierten Motorenapplikation durch die neue CAMEO-Online-DoE-Toolbox," ATZ *Automobiltechnische Zeitschrift*, July/August 2001, pp. 636–643.

[30] Roy, Ranjit K., *Design of Experiments Using the Taguchi Approach. Steps to Product and Process Improvement*, John Wiley & Sons, New York, 2001.

[31] Montgomery, Douglas, *Design and Analysis of Experiments*, John Wiley & Sons, New York, 2001.

[32] Deutsches Institut für Normung e.V., DIN 19226-1—Leittechnik; *Regelungstechnik und Steuerungstechnik*, Allgemeine Grundbegriffe, February 1994.

[33] Föllinger, Otto, *Regelungstechnik. Einführung in die Methoden und ihre Anwendung*, Hüthig-Verlag, Heidelberg, Germany, 1994.

[34] Unbehauen, Heinz, *Regelungstechnik*, Vols. 1–3, Vieweg-Verlag, Wiesbaden, Germany, 2000.

[35] Kiencke, Uwe, and Nielsen, Lars, *Automotive Control Systems. For Engine, Driveline, and Vehicle*, Springer-Verlag, Berlin, Heidelberg, New York, 2000.

[36] Mayr, Robert, *Regelungsstrategien für die automatische Fahrzeugführung. Längs- und Querregelung, Spurwechsel- und Überholmanöver*, Springer-Verlag, Berlin, Heidelberg, New York, 2001.

[37] Kiencke, Uwe, *Signale und Systeme*, R. Oldenbourg Verlag, Munich, Vienna, 1998.

[38] Kiencke, Uwe, *Ereignisdiskrete Systeme. Modellierung und Steuerung verteilter Systeme*, R. Oldenbourg Verlag, Munich, Vienna, 1997.

[39] Robert Bosch GmbH (Ed.), *Mikroelektronik im Kraftfahrzeug*, Robert Bosch GmbH, Stuttgart, Germany, 2001.

[40] Robert Bosch GmbH (Ed.), *Sensoren im Kraftfahrzeug*, Robert Bosch GmbH, Stuttgart, Germany, 2001.

[41] Robert Bosch GmbH (Ed.), *Autoelektrik/Autoelektronik, Systeme und Komponenten*,

4th Edition, Vieweg-Verlag, Wiesbaden, Germany, 2002.

[42] Liu, Jane W.S., *Real-Time Systems*, Prentice-Hall, Upper Saddle River, NJ, 2000.

[43] Wettstein, H., *Architektur von Betriebssystemen*, 3rd Edition, Carl Hanser Verlag, Munich, Germany, 1987.

[44] International Telecommunication Union, Message Sequence Charts, ITU-T Recommendation Z. 120, Geneva, Switzerland, 1994.

[45] Kopetz, Hermann, *Real-Time Systems. Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, Norwell, MA, 2002.

[46] International Organization for Standardization/International Electrotechnical Commission, ISO/IEC 7498: Informationstechnik—Kommunikation Offener Systeme—Basis-Referenzmodell, 1994.

[47] International Organization for Standardization/International Electrotechnical Commission, ISO/IEC 10731: Informationstechnik—Kommunikation Offener Systeme—Basis-Referenzmodell—Konventionen für Definition von OSI-Diensten, 1995.

[48] Etschberger, Konrad, *Controller Area Network. Grundlagen, Protokolle, Bausteine, Anwendungen*, 3rd Edition, Carl Hanser Verlag, Munich, Vienna, 2002.

[49] FlexRay, www.flexray.com.

[50] TTP Time Triggered Protocol, www.tttech.com.

[51] International Organization for Standardization, ISO 11898-4: Time Triggered CAN, 2002.

[52] Isermann, Rolf (Ed.), *Überwachung und Fehlerdiagnose. Moderne Methoden und ihre Anwendungen bei technischen Systemen*, VDI-Verlag, Düsseldorf, Germany, 1994.

[53] Birolini, Alessandro, *Reliability Engineering. Theory and Practice*, Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[54] Birolini, Alessandro, *Zuverlässigkeit von Geräten und Systemen*, Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[55] Ehrenberger, Wolfgang, *Software-Verifikation: Verfahren für den Zuverlässigkeitsnachweis von Software*, Carl Hanser Verlag, Munich, Vienna, 2002.

[56] U.S. Environmental Protection Agency, Control of Air Pollution from Motor Vehicles and New Motor Vehicles; Modification of Federal On-Board Diagnostic Regulations for Light-Duty Vehicles and Light-Duty Trucks; Extension of Acceptance of California OBD II Requirements, December 1998.

[57] Halang, W.A., and Konakovsky, R., *Sicherheitsgerichtete Echtzeitsysteme*, R. Oldenbourg Verlag, Munich, Vienna, 1999.

[58] Lin, Shu, and Costello, Daniel J., *Error Control Coding*, Prentice-Hall, Englewood Cliffs,

NJ, 1983.

[59] Leveson, Nancy G., *Safeware. System Safety and Computers. A Guide to Preventing Accidents and Losses Caused by Technology*, Addison-Wesley, New York, 1995.

[60] Storey, Neil, *Safety-Critical Computer Systems*, Prentice-Hall, Harlow, England, 1996.

[61] Deutsches Institut für Normung e.V., DIN 25448—Ausfalleffektanalyse (Fehler-Möglichkeits- und Einfluss-Analyse), May 1990.

[62] "Der neue BMW 7er," *Automobiltechnische Zeitschrift/Motortechnische Zeitschrift*, ATZ/MTZ Extra (special issue), November 2001.

[63] International Organization for Standardization, ISO 11519: Straßenfahrzeuge—Serielle Datenübertragung mit niedriger Übertragungsrate, 1994.

[64] Byteflight, www.byteflight.de.

[65] MOST Media Orientated System Transport, www.mostcooperation.com.

[66] "Die neue Mercedes-Benz-E-Klasse," *Automobiltechnische Zeitschrift/Motortechnische Zeitschrift*, ATZ/MTZ Extra (special issue), May 2002.

[67] Bluetooth, www.bluetooth.com.

[68] LIN Local Interconnect Network, www.lin-subbus.de.

[69] Boy, J., Dudek, C., and Kuschel, S., *Projektmanagement. Grundlagen, Methoden und Techniken, Zusammenhänge*, Gabal-Verlag, Offenbach, Germany, 1998.

[70] The Motor Industry Software Reliability Association, Development Guidelines for Vehicle Based Software, 1994, http://www.misra.org.uk.

[71] International Council on Systems Engineering, http://www.incose.org.

[72] Balzert, Helmut, *Lehrbuch der Software-Technik*, 2nd Edition, Spektrum Akademischer Verlag, Heidelberg—Berlin, Germany, 2000.

[73] ETAS GmbH (Ed.), ASCET V5.0 User's Guide. ETAS GmbH, Stuttgart, Germany, 2004.

[74] Selic, B., Gullekson, G., and Ward, P.T., *Real-Time Object-Oriented Modeling*, John Wiley & Sons, New York, 1994.

[75] Stoer, Josef, *Numerische Mathematik 1*, 8th Edition, Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[76] Kernighan, B.W., and Ritchie, D.M., *Programmieren in C*, 2nd Edition, Carl Hanser Verlag, Munich, Germany, 1990.

[77] Broy, Manfred, *Informatik. Eine grundlegende Einführung, Vols. 1/2*, Springer-Verlag, Berlin, Heidelberg, New York, 1998.

[78] Wirth, N., *Grundlagen und Techniken des Compilerbaus*, Addison-Wesley, Bonn, Germany, and Paris, France, 1996.

[79] Manufacturer Supplier Relationship, Working Groups MEGMA and MEDOC, http://www.msr-wg.de.

[80] The Motor Industry Software Reliability Association, Guidelines for the Use of the C Language in Vehicle-Based Software, 1998.

[81] van Basshuysen, Richard, and Schäfer, Fred (Eds.), *Handbuch Verbrennungsmotor. Grundlagen, Komponenten, Systeme, Perspektiven*, 1st Edition, Vieweg-Verlag, Wiesbaden, Germany, 2002; also published as van Basshuysen, Richard, and Schäfer, Fred (Eds.), *Internal Combustion Engine Reference Handbook*, SAE International, Warrendale, PA, 2004.

[82] Pauli, B., and Meyna, A., "Zuverlässigkeitsprognosen für elektronische Steuergeräte im Kraftfahrzeug" ("Reliability of Electronic Control Units in Motor Vehicles"; orginal German), in Proceedings of the 7th International Congress "Electronic Systems for Vehicles," Baden-Baden, Germany, September 13, 1996, ed. by VDI Society Vehicle and Transportation Technology, VDI-Berichte 1287, VDI-Verlag, Düsseldorf, Germany, 1996, pp. 87–105.

[83] Beer, A., and Schmidt, M., "Funktionale Sicherheit Sicherheitsrelevanter Systeme im Kraftfahrzeug" ("Functional Safety of Safety Relevant Systems in Vehicles"; original German), in Proceedings of the 9th International Congress "Electronic Systems for Vehicles," Baden-Baden, Germany, October 5–6, 2000, ed. by VDI Society Vehicle and Transportation Technology, VDI-Berichte 1547, VDI-Verlag, Düsseldorf, Germany, 2000, pp. 391–409.

[84] UML Unified Modeling Language™, www.uml.org.

[85] Harel, D., *Statecharts. A Visual Formalism for Complex Systems, Science of Computer Programming, Vol. 8*, Elsevier Science Publishers, North Holland, 1987.

[86] ETAS GmbH (Ed.), ERCOS^{EK} V4.2 User's Guide, ETAS GmbH, Stuttgart, Germany, 2002.

[87] ETAS GmbH (Ed.), INCA V5.0 User's Guide, ETAS GmbH, Stuttgart, Germany, 2004.

[88] Grams, T., *Denkfallen und Programmierfehler*, Springer-Verlag, Berlin, Heidelberg, New York, 1990.

[89] ETAS GmbH (Ed.), Data Declaration System V2.3 User's Guide, ETAS GmbH, Stuttgart, Germany, 2001.

[90] ETAS GmbH (Ed.), LABCAR-OPERATOR V2.0 User's Guide, ETAS GmbH, Stuttgart, Germany, 2003.

[91] Kühner, T., Seefried, V., Litschel, M., and Schelling, H., "Realisierung Virtueller Fahrzeugfunktionen für Vernetzte Systeme auf Basis Standardisierter Software-Bausteine" ("Implementation of Virtual Vehicle Functions for Networked Systems Using Standardized

Software Modules"; original German), in Proceedings of the 7th International Congress "Electronic Systems for Vehicles," Baden-Baden, Germany, September 12–13, 1996, ed. by VDI Society Vehicle and Transportation Technology, VDI-Berichte 1287, VDI-Verlag, Düsseldorf, Germany, 1996, pp. 691–708.

[92] Institute of Electrical and Electronics Engineers: NEXUS, www.ieee-isto.org/Nexus5001.

[93] Institute of Electrical and Electronics Engineers: JTAG IEEE 1149.1, www.ieee.org.

[94] ETAS GmbH (Ed.), ETK S2.0 Emulator Probe for Serial Debug Interfaces, Data Sheet, ETAS GmbH, Stuttgart, Germany, 2002.

[95] ETAS GmbH (Ed.), ETK 7.1 16-Bit Emulator Probe, Data Sheet, ETAS GmbH, Stuttgart, Germany, 2001.

[96] Gumpinger, F., Huber, F.-M., and Siefermann, O., "BMW Car & Key Memory: Der Kunde bekommt sein individuelles Fahrzeug" ("BMW Car & Key Memory—The Customer Will Receive His Individual Vehicle"; original German), in Proceedings of the 8th International Congress "Electronic Systems for Vehicles," Baden-Baden, Germany, October 8–9, 1998, ed. by VDI Society Vehicle and Transportation Technology, VDI-Berichte 1415, VDI-Verlag, Düsseldorf, Germany, 1998, pp. 995–1007.

[97] Singh, S., *Geheime Botschaften. Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet*, Deutscher Taschenbuch Verlag, Munich, Germany, 2001; also published as Singh, S., *The Code Book, Fourth Estate*, ISBN 1857028899, 2000.